**Firebird:** Conheça e saiba como e quando utilizar diretamente as tabelas de sistema do FB

# Clube Delphi +PHP

DevMedia group

**Boas Práticas**
Construa aplicações que suportam extensão e modularização com Plugins

**Delphi for PHP**
Integração com Google Maps, Segurança, Banco de Dados e JavaScript

Ano 9 :: Edição 115 :: R$ 11,90

ISSN 1517990-7

9 771517 990009 00115

# Mapeamento Objeto / Relacional

**Brinde**

**Comprando esta edição ganhe 4 vídeo-aulas:**

- Desenvolvimento Web vs. Desktop: passando dados entre formulários
- Atualizador automático de sistemas
- Criando gráficos com DBChart
- Usando a linguagem nativa do Rave Reports para incluir lógica nos relatórios

# MELHOR QUE SERVIDOR DEDICADO
# CLOUD SERVER REDEHOST

A partir de R$ 99,00 mensais

O Cloud Server da RedeHost é a solução com melhor relação custo/benefício para aplicações que necessitem de um servidor dedicado. Por ser desenvolvido sobre conceitos de Cloud Computing utiliza da alta disponibilidade oferecida pela "nuvem computacional" com flexibilidade de expandir a capacidade de seu servidor conforme a necessidade de seu negócio.

## ESCALABILIDADE

Aumente os recursos de processamento, memória, disco, e banda quando desejar.

## DISPONIBILIDADE

Em caso de falha os recursos são automaticamente realocados.

## PERFORMANCE

Memória, disco e banda 100% garantidos.

## MENOR CUSTO

Melhor que um servidor dedicado com menor custo.

www.redehost.com.br

| | | |
|---|---|---|
| SP 11 4062.0909 | RJ 21 4062.0909 | MG 31 4062.0909 |
| RS 51 4062.0909 | SC 48 4062.0909 | PR 41 4062.0909 |

RedeHost

# Sumário


Quick Update

**8 –DataSetProvider e BeforeUpdateRecord**
[ *Guinther Pauli* ]

Delphi — Banco de Dados

**12 – Firebird**
Manipulando tabelas de sistema
[ *Daniel Sobrinho Laporte* ]

Boas Práticas — Boas Práticas

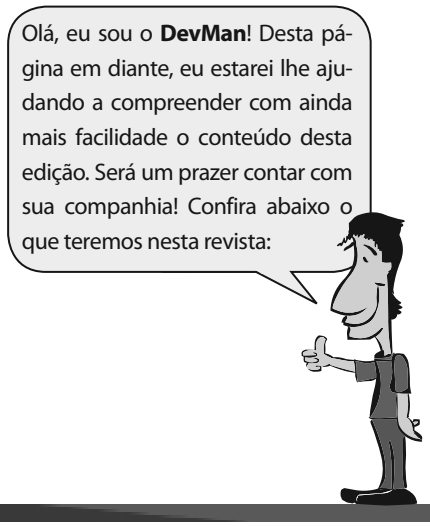
**20 – Persistência de Objetos**
Usando o Delphi com o mais popular e poderoso framework de Mapeamento Objeto / Relacional
[ *Paulo Quicoli* ]

Boas Práticas — Boas Práticas, Boa Ideia

**36 – Desenvolvendo aplicativos com suporte a Plugins**
Aprenda como modificar sistemas sem alterar uma única linha de código
[ *Rafael Stavarengo* ]

PHP — Delphi for PHP

**44 – Dicas de Delphi for PHP**
Google Maps, Segurança, Banco de Dados e JavaScript
[ *Rafael Silva* ]


## Brinde na web desta edição - 4 vídeos! Confira:

4 Vídeos

- **Rave Language**
*http://www.devmedia.com.br/articles/viewcomp.asp?comp=15486*

- **Atualizador Automático**
*http://www.devmedia.com.br/articles/viewcomp.asp?comp=15056*

- **Utilizando o DBChart para criação de gráficos**
*http://www.devmedia.com.br/articles/viewcomp.asp?comp=15566*

- **Comparativo VCL x ASP.NET com Delphi Prism - Passando dados entre forms**
*http://www.devmedia.com.br/articles/viewcomp.asp?comp=15565*

Para acessar os vídeos utilize os seguintes dados
*Login:* **DVM.PL** *Senha:* **alskdj**


**Gostou das vídeo aulas?** O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

## Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:
*www.devmedia.com.br/clubedelphi/feedback*

Para votar, você vai precisar do código e senha de banca desta edição, que é:
Login: **DVM.PL** Senha: **alskdj**

Clube Delphi +PHP

Ano 9 - 115ª Edição 2010 - ISSN 1517990-7 Impresso no Brasil

**Corpo Editorial**

**Editor Geral**
Guinther Pauli
guinther@devmedia.com.br

**Editor Técnico**
Paulo Quicoli
pauloquicoli@devmedia.com.br

Daniel Laporte
daniel.laporte@gmail.com

Rodrigo Carreiro Mourão
rodrigocarreiro@rmfactory.com.br

**Equipe Editorial**
Vitor Luiz Rubio
vitorrubio@gmail.com

Rafael Stavarengo
faelsta@gmail.com

**Capa e Diagramação**
Romulo Araujo
romulo@devmedia.com.br

**Coordenação Geral**
Daniella Costa - daniella@devmedia.com.br

**Revisão e Supervisão**
Thiago Vincenzo - thiago.v.ciancio@devmedia.com.br

**Distribuição**
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

**Atendimento ao Leitor**

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

**Cristiany Queiroz** – Atendimento ao Leitor
*www.devmedia.com.br/mancad*
(21) 3382-5038

**Kaline Dolabella –** Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5038

**Publicidade**

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

**Kaline Dolabella**
publicidade@devmedia.com.br

**Fale com o Editor!**

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

**Guinther Pauli - Editor da Revista**
*guinther@devmedia.com.br*

## EDITORIAL

Para os desenvolvedores Delphi, mapeamento objeto relacional (Object-Relational Mapping -ORM) e persistência de dados nunca foi um problema, aliás, convivemos durante anos sem nem mesmo ouvir esses termos. Nosso framework de "persistência" se chama dbExpress, aliado ao ClientDataSet. Mapeamento? Responda-me sinceramente prezado amigo leitor, você já criou uma classe chamada "Cliente"? Provavelmente a maioria dirá que não. Para que é preciso criar uma tabela chamada Cliente no banco, depois uma classe chamada Cliente, se simplesmente colocamos no DataModule um TClientDataSet que manipula diretamente esses dados? Sinceramente, eu não vejo utilidade nenhuma um desenvolvedor Delphi criar um diagrama de classes (domínio) representando o que seria o modelo ER (Entidade Relacionamento) do banco de dados. A menos que você utilize um framework de mapeamento objeto/relacional, de persistência, ou construa o seu próprio. Aí a história muda.

Ao utilizar um framework de persistência, passamos a usar boas práticas de desenvolvimento e programação orientada a objetos. Deixamos a SQL de lado, passamos a criar objetos ao invés de inserir registros. Passamos a nos concentrar mais nos problemas dos nossos clientes, que é o que realmente importa, do que nos problemas de infraestrutura da aplicação. Deixamos muitos dos detalhes inerentes à comunicação com o banco de dados por conta do framework. Passamos a aplicar em situações reais poderosos padrões de desenvolvimento.

O Hibernate é o framework de mapeamento objeto relacional mais popular e poderoso do mercado. Ele foi originalmente desenvolvido para a plataforma Java. Ganhou adeptos e uma grande comunidade ao redor do mundo. É um framework robusto, consolidado, maduro, muito bem testado. E a melhor parte: o Hibernate ganhou uma versão para o .NET, chamada intuitivamente de NHibernate. Com isso, tudo o que já existia de bom e de melhor no Hibernate ficou ao alcance de uma outra grande comunidade de desenvolvedores, o que inclui nós desenvolvedores Delphi. Nesta edição, Paulo Quicoli traz um excelente artigo sobre NHibernate com Delphi.

Ainda nesta edição, veja como trabalhar com tabelas de sistemas do Firebird. Também aprenda a criar sistemas que suportam plugins, como a maioria dos softwares profissionais existentes no mercado atualmente. Na sessão PHP, excelentes dicas de Delphi for PHP para os desenvolvedores Web.

Boa leitura!

**Guinther Pauli**

*guinther@devmedia.com.br*

Certificado Delphi 3,5,6,7,2005,2006,Web,Kylix

Microsoft Certified: MCP,MCAD,MCSD.NET,MCTS,MCPD (ASP.NET,C#)

guintherpauli.blogspot.com

twitter.com/guintherpauli

ClubeDelphi PLUS

DevMedia group

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
*http://www.devmedia.com.br/clubedelphi/portal.asp*

# Perguntas e Respostas

**Perguntas respondidas por:**
*Guinther Pauli: guinther.pauli@gmail.com*

*Estou com um problema para ser resolvido, no que diz respeito à integração e troca de dados entre diferentes plataformas usando XML, neste caso, uma aplicação.NET e uma aplicação cliente Delphi Win32. Quando tento abrir o XML que recebo, usando o ClientDataSet, recebo a mensagem "Mismatch in DataPacket". Como posso ler estes dados XML em um ClientDataSet?Por que os dados gerados em XML pela aplicação .NET não são suportados pelo Delphi?*

*Nigor Macedo Feltrin*
*Sorriso - MT*

Vamos fazer um exemplo aqui usando Delphi Prism (.NET) e Delphi 2010 (Win32) que vai demonstrar exatamente como realizar a interoperabilidade necessária e resolver o seu problema. O exemplo consistirá da seguinte arquitetura: uma aplicação servidora, neste caso um Web Service, que se conectará em um banco de dados SQL Server 2008 via ADO.NET e vai extrair dados de uma tabela de produtos. A seguir, estes dados poderão ser consultados a partir da Web a partir de qualquer aplicação cliente que suporte XML. Nesse caso, nossa aplicação cliente será escrita em Delphi 2010.

No Delphi Prism, criamos um novo ASP.NET Web Service. O código da classe que faz acesso ao banco de dados pode ser vista na **Listagem 1**. Aqui estamos consultando uma tabela chamada Products do banco de dados de demonstração do SQL Server, chamado Northwind. Basicamente usamos código ADO.NET para acessar o banco. A parte mais interessante vem no final do método, veja que não devolvemos o tradicional "DataSet", classe do ADO.NET utilizada para troca de dados em XML. Ao invés disso, estamos devolvendo um XmlDocument. O método retorna os produtos de uma categoria passada por parâmetro. A **Listagem 2** mostra a entrada da string de conexão no arquivo Web.Config.

Executando o exemplo, teremos uma página de teste onde podemos informar o código de uma categoria ("1" por exemplo) e obter os produtos relacionados, em formato XML (**Figura 1**). Salve este documento XML em uma pasta qualquer. Guarde também o endereço do Web Service de teste, neste caso http://localhost:5129/Server/Service.asmx, pois o utilizaremos adiante.

**Nota**

Para reproduzir este exemplo, tenha instalado em sua máquina o banco Northwind, disponível no endereço tinyurl.com/northwinddb. Ele deve ser registrado (Attach) no SQL Server Express, que acompanha o Prism, usando o SQL Server Management Studio Express. Note que você também pode usar o Firebird e o seu provider nativo ADO.NET para criar uma arquitetura idêntica.

**Listagem 1**. Código do Web Service em Delphi Prism

```
namespace;

interface
uses
  System,
  System.Web,
  System.Collections,
  System.Web.Services,
  System.Web.Services.Protocols,
  System.Data,
  System.Data.SqlClient,
  System.Xml,
  System.Web.Configuration,
  System.IO;

type
  [WebService(&Namespace := 'http://tempuri.org/')]
  [WebServiceBinding(ConformsTo := WsiProfiles.BasicProfile1_1)]
  Service = public class(System.Web.Services.WebService)
  public
    [WebMethod]
    method GetProducts(CategoryId: Integer): XmlDocument;
  end;

implementation

method Service.GetProducts(CategoryId: Integer): XmlDocument;
begin
  var ConStr := WebConfigurationManager.ConnectionStrings
    ["NW"].ConnectionString;
  var con := new SqlConnection(ConStr);
  var SQL := "select * from Products where CategoryId=@
   CategoryId";
  var cmd := new SqlCommand(SQL,con);
  cmd.Parameters.AddWithValue("@CategoryId",CategoryId);
  var da := new SqlDataAdapter(cmd);
  var dt := new DataTable("Products");
  da.Fill(dt);
  var sw := new StringWriter();
  dt.WriteXml(sw);
  var doc := new XmlDocument();
  doc.LoadXml(sw.ToString());
  result := doc;
end;
end.
```

**Listagem 2**. ConnectionString

```
<connectionStrings>
  <add name="NW" connectionString="Data Source=.\
    SQLEXPRESS;Initial Catalog=NORTHWIND;Integrated Security=true"/>
</connectionStrings>
```

**Nota**

Veja que, como estamos usando o servidor Web de testes do Prism e não o IIS, o número da porta especificada na URL vai variar.

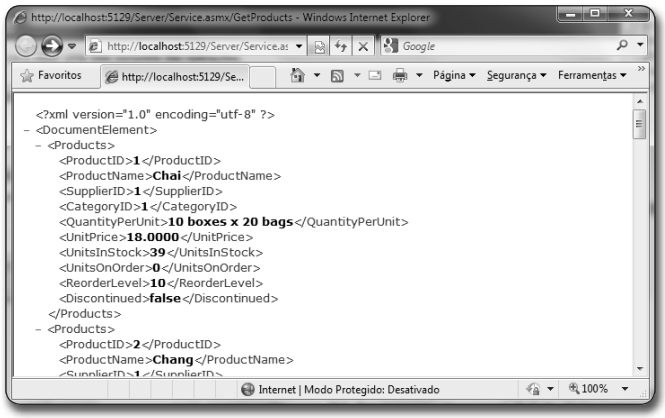



**Figura 1.** Web Service devolve documento XML com dados de uma tabela

Se você tentar abrir neste momento o arquivo XML gerado pelo Web Service em um ClientDataSet no Delphi, obterá o erro da **Figura 2**, o mesmo erro que reportou em sua pergunta. Isso acontece porque o ClientDataSet trabalha com um formato mais específico de XML, o que chamamos de DataPacket. Para permitir essa integração, precisamos criar um arquivo que transforme o XML gerado pelo .NET em um XML DataPacket. Isso é tarefa para o XML Mapper do Delphi.

**Figura 2.** XML gerado no Web Service não é um DataPacket

No XML Mapper, clique em Open e localize o arquivo XML salvo no disco. Na coluna da esquerda, dê um duplo clique nos campos que deseja transformar. A seguir clique em Create > DataPacket from XML, depois em Create and Test Transformation e por fim salve o arquivo de transformação em File > Save > Transformation. A tela do seu XML Mapper deve estar semelhante à **Figura 3**.

No Delphi 2010, iniciamos uma nova aplicação Win32, desenhando a tela mostrada na **Figura 4**. Aqui temos um PageControl com duas abas, uma com um Memo (que vai mostrar o XML "puro") e outra com um DBGrid (que vai mostrar os dados transformados em DataPacket). Temos um XmlDocument que vai fazer a leitura dos dados do Web Service e armazenar em memória. Para que esses dados em XML sejam transformados em DataPacket, temos o XMLTransformProvider, um componente da mesma família do DataSetProvider que está indicado na propriedade ProviderName de um ClientDataSet. Na sua propriedade TransformRead, indicamos o arquivo de transformação gerado pelo Mapper. Para facilitar a construção do form, veja seu código DFM sintetizado na **Listagem 3**.

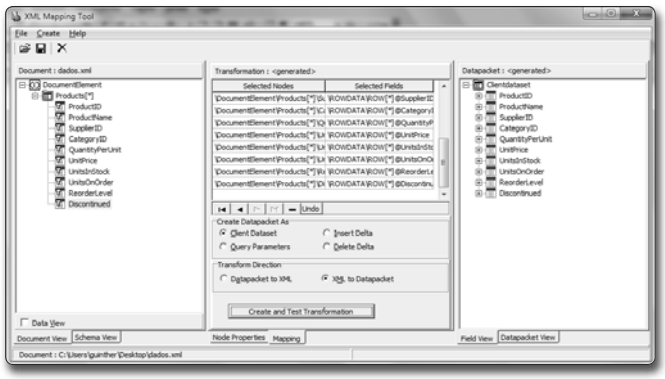



**Figura 3.** XML Mapper

**Listagem 3**. DFM do formulário

```
object FrmMain: TFrmMain
  object PageControl1: TPageControl
    object TabSheet1: TTabSheet
      Caption = 'Texto'
      object Memo1: TMemo
      end
    end
    object ClientDataSet: TTabSheet
      Caption = 'ClientDataSet'
      object DBGrid1: TDBGrid
        DataSource = DataSource1
      end
    end
  end
  object Edit1: TEdit
  end
  object Button1: TButton
    OnClick = Button1Click
  end
  object XMLDocument1: TXMLDocument
    Active = True
    DOMVendorDesc = 'MSXML'
  end
  object ClientDataSet1: TClientDataSet
    ProviderName = 'XMLTransformProvider1'
  end
  object XMLTransformProvider1: TXMLTransformProvider
    TransformRead.TransformationFile = 'ToDp.xtr'
  end
  object DataSource1: TDataSource
    DataSet = ClientDataSet1
  end
end
```

Antes de codificarmos o botão que fará o acesso ao Web Service, precisamos ir até o menu File > New > Other > WebServices > WSDL Importer e importar a WSDL do serviço, informando o endereço previamente salvo. Esse procedimento fará com que seja gerada uma classe proxy que permite acesso ao serviço. Internamente esse proxy usa um componente HTTPRIO.

O código do botão pode ser visto na **Listagem 4**. Aqui chamamos o método remoto usando SOAP, passando como parâmetro o ID da categoria informado no Edit. Recebemos então um XML que é transformado pelo XMLTransformProvider gerando um DataPacket, armazenado no ClientDataSet e exibido no DBGrid (**Figura 5**).

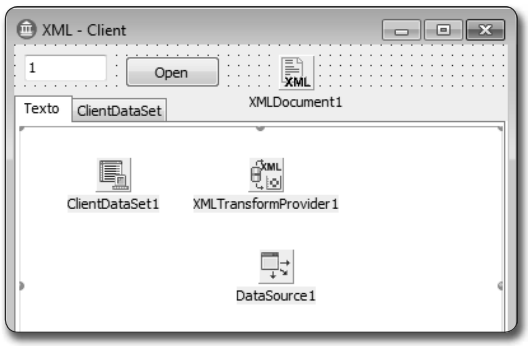
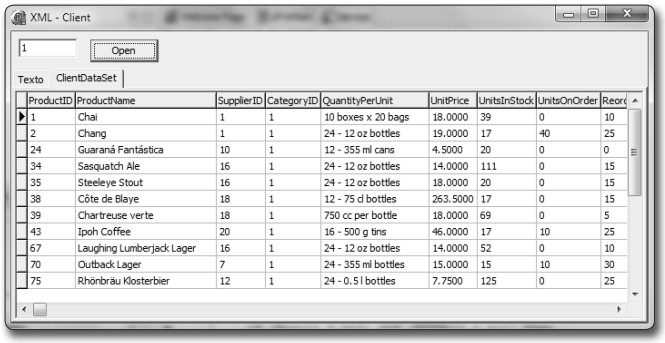



**Figura 4.** Aplicação Win32 que irá acessar o Web Service




| ProductID | ProductName | SupplierID | CategoryID | QuantityPerUnit | UnitPrice | UnitsInStock | UnitsOnOrder | Reord |
|---|---|---|---|---|---|---|---|---|
| 1 | Chai | 1 | 1 | 10 boxes x 20 bags | 18.0000 | 39 | 0 | 10 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19.0000 | 17 | 40 | 25 |
| 24 | Guaraná Fantástica | 10 | 1 | 12 - 355 ml cans | 4.5000 | 20 | 0 | 0 |
| 34 | Sasquatch Ale | 16 | 1 | 24 - 12 oz bottles | 14.0000 | 111 | 0 | 15 |
| 35 | Steeleye Stout | 16 | 1 | 24 - 12 oz bottles | 18.0000 | 20 | 0 | 15 |
| 38 | Côte de Blaye | 18 | 1 | 12 - 75 cl bottles | 263.5000 | 17 | 0 | 15 |
| 39 | Chartreuse verte | 18 | 1 | 750 cc per bottle | 18.0000 | 69 | 0 | 5 |
| 43 | Ipoh Coffee | 20 | 1 | 16 - 500 g tins | 46.0000 | 17 | 10 | 25 |
| 67 | Laughing Lumberjack Lager | 16 | 1 | 24 - 12 oz bottles | 14.0000 | 52 | 0 | 10 |
| 70 | Outback Lager | 7 | 1 | 24 - 355 ml bottles | 15.0000 | 15 | 10 | 30 |
| 75 | Rhönbräu Klosterbier | 12 | 1 | 24 - 0.5 l bottles | 7.7500 | 125 | 0 | 25 |

**Figura 5**. Resultado da chamada ao Web Service – dados em tempo real

Note que o arquivo XML que salvamos serviu apenas para criar a transformação. A partir de agora, os dados que vemos no DBGrid são dados obtidos em tempo real a partir do Web Service. Faça um teste, altere um campo da tabela no SQL Server e veja que após a consulta os dados são automaticamente refletidos no DBGrid. Lembre-se também de que o XML Mapper permite criar o mapeamento inverso, ou seja, você pode editar os dados e enviá-los de volta ao servidor para atualização. ●

**Listagem 4**. Acessando o Web Service

```
...
uses Service;

procedure TFrmMain.Button1Click(Sender: TObject);
var
  Cat: integer;
  Doc: GetProductsResult;
begin
  Cat := StrToInt(Edit1.Text);
  Doc := GetServiceSoap().GetProducts(Cat);
  XMLDocument1.DocumentElement := Doc.XMLNode;
  Memo1.Lines.Clear();
  Memo1.Lines.Add(XMLDocument1.XML.Text);
  ClientDataSet1.Close();
  XMLTransformProvider1.TransformRead.SourceXmlDocument :=
   XMLDocument1.DOMDocument;
  ClientDataSet1.Open();
end;
```

Questão respondida pelo editor Guinther Pauli
Envie suas dúvidas para nossa equipe de editores:
*guinther@devmedia.com.br*
*pauloquicoli@devmedia.com.br*
*daniel.laporte@devmedia.com.br*
*rodrigo.carreiro@devmedia.com.br*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

# Quick Update:
# DataSetProvider e BeforeUpdateRecord

**Guinther Pauli**
*guinther.pauli@gmail.com*
*http://gpauli.com*
*http://twitter.com/guintherpauli*
*Tem 22 anos de experiência na área de desenvolvimento e arquitetura de software. Certificado em Delphi 3, 5, 6, 7, 2005, 2006, Delphi para Web e Kylix, certificado Microsoft MCP, MCAD, MCSD.NET, MCTS e MCPD (C#, ASP.NET). Editor da Revista ClubeDelphi e da Revista .NET Magazine Brasil, professor de Pós-Graduação da UNIVEM-SP. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, atuando também em projetos de migração orientando empresas e desenvolvedores na adotação de melhores práticas. Foi arquiteto e desenvolvedor do site de doações on-line do projeto Criança Esperança da Rede Globo / Unesco.*

Neste QuickUpdate vamos falar de um importante evento presente em um dos principais componentes de acesso a dados do Delphi, o evento BeforeUpdateRecord do DataSetProvider. Em primeiro lugar, vamos examinar como ele funciona.

Sempre que o usuário faz alterações nos dados de um ClientDataSet, por exemplo, alterando valores em um DBGrid, essas atualizações vão ficando em memória (cache) até que o programador chame em algum momento o método ApplyUpdates. Essa "cache" se chama Delta, e muitas vezes é composta de vários registros que vão ser atualizados em um único processamento. Ao chamar ApplyUpdates, o Delta é repassado ao DataSetProvider, que vai tentar atualizar automaticamente a tabela de origem baseado no DataSet associado. Para isso, ele faz parse do código SQL que você digitou, por exemplo, em um SQLDataSet do dbExpress. Ele faz isso através de uma interface chamada IProviderSupport, justamente para não se prender a uma arquitetura de acesso em específico e poder desfrutar de boas práticas de programação orientada a objetos, como programação para interfaces e abstração.

O DataSetProvider recebe então esse Delta e, registro a registro, tenta aplicar os dados no banco de dados de origem, gerando instruções de Insert, Update e Delete automaticamente. Porém, o DataSetProvider te dá uma última chance de manipular os dados a serem atualizados antes de enviá-los ao BD, isso acontece justamente no evento BeforeUpdateRecord. É nele, por exemplo, que podemos fazer validações, consistências mais dinâmicas, alterar valores, gerar valores default, criptografar campos, processar

dados no AppServer no caso de soluções multicamadas, fazer um backup ou log dos dados alterados, enviar dados para outro banco, iniciar uma transação explicitamente, enfim, as possibilidades são muitas.

Uma situação que é bastante comum de acontecer, e que só pode ser resolvida pela manipulação deste evento, consiste quando temos uma SELECT que faz JOIN em duas tabelas. Normalmente, o usuário atualiza os dados em tela e o DataSetProvider vai fazer UPDATE em um única tabela, apesar do JOIN. Por exemplo, se o usuário alterou o Departamento do Funcionário, não precisamos fazer um UPDATE na tabela de Departamentos, mas sim apenas atualizar a FK que vincula o Funcionário ao Departamento (ID_Departamento).

Porém, existe uma situação típica em que precisamos atualizar, em um único UPDATE, duas ou mais tabelas, normalmente quando temos relacionamentos 1-1 para simular uma "herança" OO no banco de dados usando Foreign Keys. Caso mais tradicional: PESSOA e ALUNO, ou PESSOA e FUNCIONARIO. Nesse caso, a segunda tabela "herda" os dados da primeira, é uma relacionamento de 1 para 1. Um funcionário é uma pessoa. Mas se o usuário alterar o nome e o cargo do funcionário, será necessário atualizar o nome da pessoa na tabela Pessoas, e o Cargo do Funcionário na tabela Funcionários.

Esta é exatamente a situação que vamos resolver aqui. Observe o diagrama ER da **Figura 1.** Aqui temos um banco de dados no SQL Server 2008. O script das tabelas e relacionamentos encontra-se na **Listagem 1**. Após criar o banco, populei as tabelas com alguns registros, como você pode ver pela **Figura 2**. Note que fiz um SELECT para trazer os dados de ambas as tabelas com um JOIN, o que também poderia ser feito com uma VIEW (aí está outro caso em que o BeforeUpdateRecord poderia ser usado).

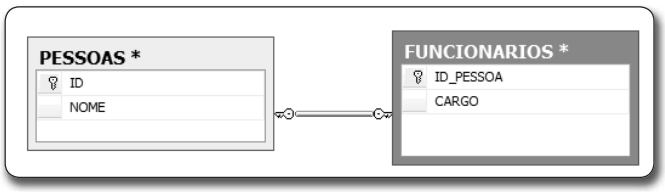



**Figura 1.** Modelo ER

**Figura 2.** Modelo ER

No Delphi, temos uma aplicação VCL Win32 tradicional, onde utilizei os componentes ADO para acesso (são mais otimizados para acesso ao SQL Server) e a dupla DataSetProvider / ClientDataSet. Um DBGrid mostra os dados. No botão, apenas chamo o ApplyUpdates. O formulário principal da aplicação pode ser visto na **Figura 2**. Para facilitar a criação do form e verificar o relacionamento entre os componentes, coloquei as principais propriedades do form na **Listagem 2**. Note as instruções SQL utilizadas e como os TFields foram configurados.

**Nota**

Neste exemplo vou utilizar SQL Server e ADO para acesso a partir do Delphi, mas nada impede de você usar dbExpress e Firebird, o procedimento é exatamente o mesmo, o DataSetProvider não depende de engine de acesso a dados e joga as mesmas regras para todos.

**Listagem 1.** Script de criação do banco de dados

```
CREATE TABLE [dbo].[PESSOAS](
 [ID] [int] IDENTITY(1,1) NOT NULL,
 [NOME] [varchar](50) NOT NULL,
 CONSTRAINT [PK_PESSOAS] PRIMARY KEY CLUSTERED
(
 [ID] ASC
)
)

GO

CREATE TABLE [dbo].[FUNCIONARIOS](
 [ID_PESSOA] [int] NOT NULL,
 [CARGO] [varchar](50) NOT NULL,
 CONSTRAINT [PK_FUNCIONARIOS] PRIMARY KEY CLUSTERED
(
 [ID_PESSOA] ASC
)
)

GO

ALTER TABLE [dbo].[FUNCIONARIOS]  WITH CHECK ADD  CONSTRAINT
[FK_FUNCIONARIOS_PESSOAS] FOREIGN KEY([ID_PESSOA])
REFERENCES [dbo].[PESSOAS] ([ID])
ON UPDATE CASCADE
ON DELETE CASCADE

GO
```

**Listagem 2.** Formulário e componentes

```
object Form1: TForm1
  object DBGrid1: TDBGrid
    DataSource = DataSource1
  object DBNavigator1: TDBNavigator
    DataSource = DataSource1
  end
  object Button1: TButton
    Caption = 'Apply'
    OnClick = Button1Click
  end
  object ADOQuery1: TADOQuery
    ConnectionString =
      'Provider=SQLNCLI.1;Integrated Security=SSPI;Persist
       Security Info=False;Initial Catalog=EMPRESA;
        DataSource=.\SQLEXPRESS'
    SQL.Strings = (
      'SELECT * FROM PESSOAS P '
      'INNER JOIN FUNCIONARIOS F'
      'ON P.ID = F.ID_PESSOA')
    object ADOQuery1ID: TAutoIncField
      FieldName = 'ID'
      ReadOnly = True
    end
    object ADOQuery1NOME: TStringField
      FieldName = 'NOME'
      Size = 50
    end
    object ADOQuery1ID_PESSOA: TIntegerField
      FieldName = 'ID_PESSOA'
    end
    object ADOQuery1CARGO: TStringField
      FieldName = 'CARGO'
      Size = 50
    end
  end
  object DataSetProvider1: TDataSetProvider
    DataSet = ADOQuery1
  end
  object ClientDataSet1: TClientDataSet
    Active = True
    ProviderName = 'DataSetProvider1'
  end
  object DataSource1: TDataSource
    DataSet = ClientDataSet1
  end
end
```

Se você executar a aplicação agora e atualizar um registro, obviamente receberá muitos erros, isso porque o DataSetProvider se perdeu totalmente. Primeiro, porque não configuramos as ProviderFlags, e nem vamos fazer isso porque não vamos atualizar uma tabela só. O erro da **Figura 3** acontece porque o DataSetProvider tentou dar um UPDATE no campo ID_PESSOA da tabela PESSOA, só que ambos sabemos que esse campo é da tabela FUNCIONARIOS.

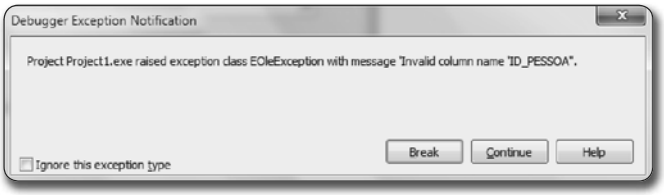



**Figura 3.** DSP tentou atualizar um campo de uma tabela em outra

O erro da **Figura 4** aconteceu porque não definimos os ProviderFlags nem o UpdateMode do DataSetProvider, logo, o DSP não conseguiu montar corretamente a cláusula Where do Update.

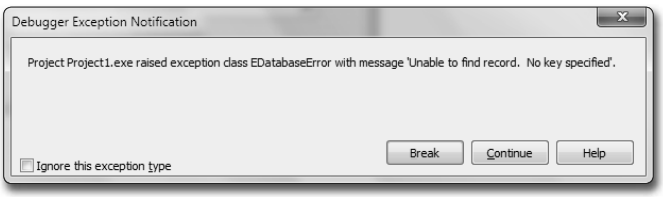



**Figura 4.** DSP precisa saber quem é a chave para gerar o update

O código da **Listagem 3** resolve o problema de forma elegante. Ele é o manipulador para o evento BeforeUpdateRecord do DSP. O que fazemos aqui é simplesmente testar se houve um Update, nesse caso, usando o TADOCommand, disparamos um UPDATE em cada tabela, separando os campos respectivos a cada uma. Por fim, dizemos ao evento através do Applied que já fizemos a atualização no "braço" é que não é para o DSP tentar fazer a sua própria atualização. Você pode fazer um teste alterando tanto o campo NOME quanto o campo CARGO no DBGrid e verificar que ambas as tabelas no banco serão atualizadas. ●

**Listagem 3.** BeforeUpdateRecord

```
procedure TForm1.DataSetProvider1BeforeUpdateRecord(Sender:
TObject;
  SourceDS: TDataSet; DeltaDS: TCustomClientDataSet; UpdateKind:
   TUpdateKind;
  var Applied: Boolean);
var
  cmd: TADOCommand;
begin
  if UpdateKind = ukModify then
  begin
    cmd := TADOCommand.Create(self);
    try
      cmd.ConnectionString := ADOQuery1.ConnectionString;
      cmd.CommandText := Format('update PESSOAS set NOME
        =%s where ID=%s',
        [QuotedStr(DeltaDS.FieldByName('NOME').AsString),
        DeltaDS.FieldByName('ID').OldValue]);
      cmd.Execute();
      cmd.CommandText := Format('update FUNCIONARIOS
        set CARGO=%S where ID_PESSOA=%s',
        [QuotedStr(DeltaDS.FieldByName('CARGO').AsString),
        DeltaDS.FieldByName('ID').OldValue]);
      cmd.Execute();
      Applied := true;
    finally
      cmd.Free();
    end;
  end;
end;
```

**Nota**

Aqui neste exemplo me preocupei apenas em tratar o Update, mas procedimento semelhante pode ser feito também para o Insert. Além disso, não me preocupei em usar transações nem parâmetros no ADO, mas lembro que esse é o procedimento ideal a ser realizado quando for colocar o código do exemplo em produção.

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:
***www.devmedia.com.br/clubedelphi/feedback***

# Chegou o Sistema de Créditos DevMedia

Agora o **conteúdo completo** do nosso site está **ao seu alcance!**

A partir de agora todas as **2000 vídeo-aulas** do site DevMedia podem ser compradas individualmente.

Economia - Você não precisa mais assinar oito produtos diferentes para ter acesso ao conteúdo completo do site!

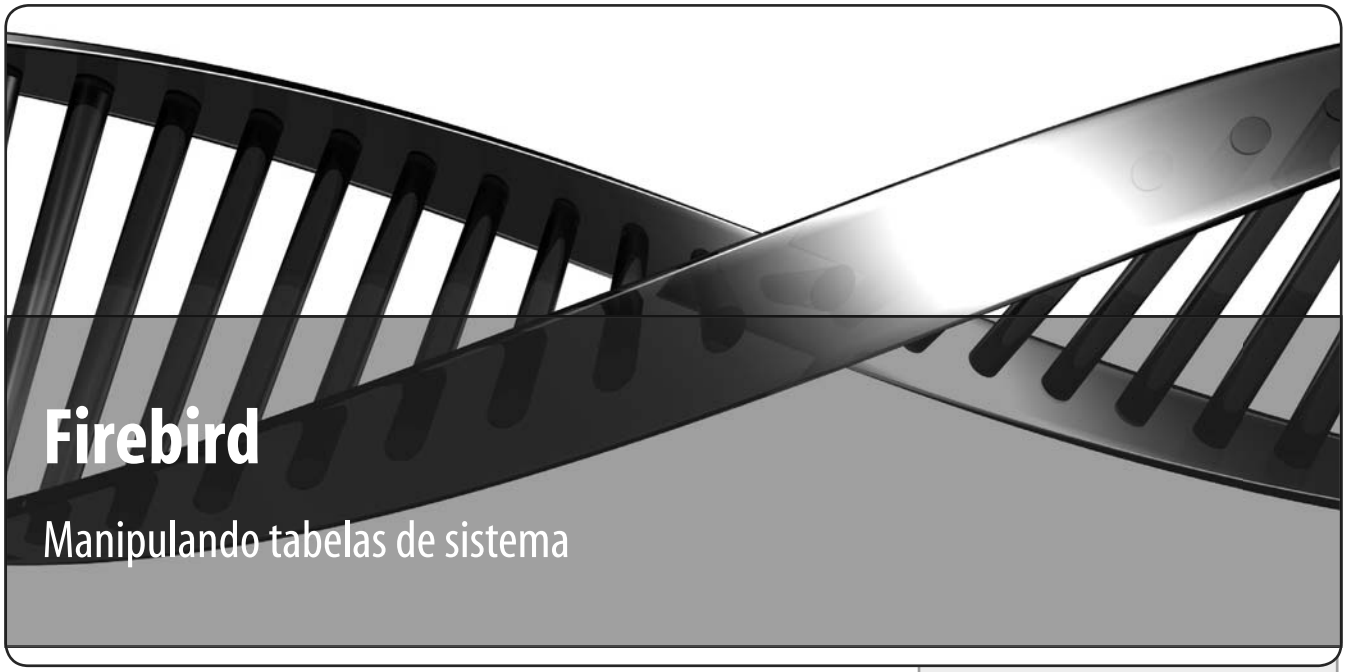
Todas as vídeos que você sempre quis ver a partir **R$ 0,75!**

Saiba mais sobre o Sistema de Créditos!

DevMedia GROUP

www.devmedia.com.br/creditos

**Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET**



# Firebird

## Manipulando tabelas de sistema

**Do que trata o artigo**
Aprofundar nos recursos avançados do banco de dados Firebird, permitindo a aplicação dos recursos de tabelas do sistema para as mais diversas situações, como mapeamento de objetos do banco, criação de recursos de validação, facilitando a manutenção.

**Para que serve**
Conhecer as tabelas de sistema do Firebird permitirá ao desenvolvedor criar aplicativos específicos para este SGBD, como utilitários, ou mesmo otimizar o acesso a partir de aplicações Delphi.

**Em que situação o tema é útil**
Através das tabelas do sistema é possível realizar tarefas das mais diversas, como criar nossas próprias aplicações de integração com o banco de dados, ou mesmo efetuar automatizações em base de dados existentes.

### Resumo do DevMan

Conhecer os recursos avançados do banco de dados de sua preferência é sempre uma boa estratégia para auxiliá-lo a criar soluções e ao mesmo tempo performáticas. Neste artigo aprenderemos sobre as tabelas do sistema que nos permitem realizar uma administração total do banco de dados mais utilizado pela comunidade Delphi.

**Daniel Sobrinho Laporte**
*daniel.sobrinho@devmedia.com.br*
*daniel.laporte@gmail.com*
*www.twitter.com/daniel_laporte*
*Editor técnico da Revista ClubeDelphi. Graduado em Sistemas de informação e pós-graduado em administração de bancos de dados pela Universidade de Franca. É analista de sistemas e desenvolvedor Delphi desde 2005, desenvolvendo aplicações cliente/servidor e multicamadas em Delphi e Firebird.*

Recentemente, em meu artigo de personalização de aplicações utilizando recursos de banco de dados (Clube Delphi 112), demonstrei uma maneira diferenciada de armazenar as configurações dos TFields, e para isso adotei a prática de utilização de algumas tabelas de sistema do Firebird. No entanto, poucos conhecem sobre as tabelas do sistema, como utilizá-las, em que situação é possível aplicá-las, quais informações estão armazenadas em cada uma e o quão útil podem ser estes recursos em suas aplicações. Deste

modo, conheceremos as tabelas de sistema, suas respectivas funcionalidades, exemplos de utilização e possíveis sugestões do que é possível fazer com este poderoso recurso presente no banco de dados.

Para isso, criaremos uma estrutura de banco de dados que possuirá uma estrutura simples, porém buscaremos criar uma gama de objetos diferenciados como Procedures, Triggers, Generators, Exceptions e Domains, permitindo assim, a exploração de grande parte destes conceitos. Durante o artigo, veremos exemplos de instruções SQL que podem nos auxiliar na obtenção de informações simples e úteis, muitas vezes necessárias no cotidiano e que às vezes por falta de conhecimento ou qualquer outro motivo, nos fazem recorrer a sites de busca ou fóruns.

Para a elaboração de nosso exemplo, trabalharemos apenas utilizando a ferramenta IBExpert para administração do banco de dados. Ao longo do artigo exibirei situações presentes no próprio software onde referenciam exemplos do que estamos trabalhando. Iremos realizar os exemplos adotando a versão 2.0 do Firebird, porém, vale lembrar que este exemplo é perfeitamente aplicável a outras versões.

## Criando o banco de dados

Como veremos, nosso banco de dados é bem simples, composto de apenas quatro tabelas, sendo elas: ENDERECO, PESSOA, DEPARTAMENTO e PESSOA_DEPARTAMENTO que nada mais é do que um relacionamento NxN (muitos para muitos) entre as tabelas PESSOA e DEPARTAMENTO. Isso se deve ao fato de que uma pessoa está diretamente relacionada a um ou mais departamentos e um departamento pode ser composto de uma ou várias pessoas. Observe a modelagem proposta na **Figura 1.**

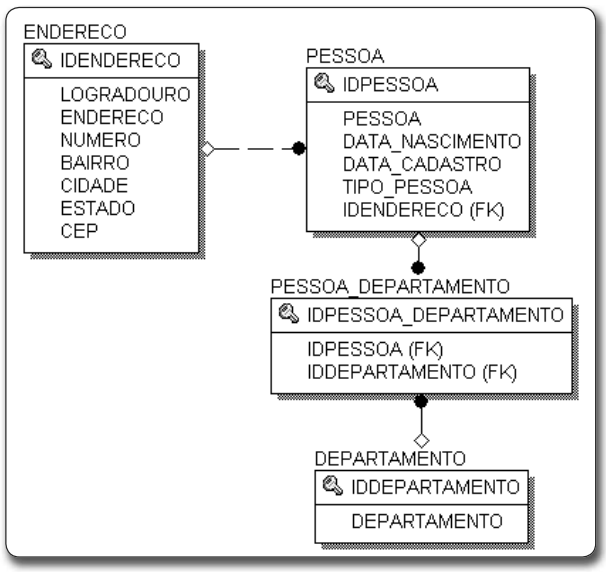



**Figura 1.** Modelagem do banco de dados

Após analisarmos a modelagem, realizaremos a criação de nosso banco de dados de exemplo. Apesar de pequeno, realizaremos a criação de alguns outros objetos que nos possibilitem a ilustração dos mais variados exemplos utilizando os recursos de tabela do sistema. Os scripts encontram-se descritos na **Listagem 1**.

Como observamos, nosso banco de dados possui estrutura simples, porém, com os objetos criados teremos a possibilidade de explorar algumas das funcionalidades das tabelas de sistema. No decorrer do artigo iremos expandir este projeto criando VIEWS, ROLES e PROCEDURES para demonstrar os demais recursos. A seguir conheceremos as tabelas do sistema e suas respectivas definições, para assim dar início a demonstração dos exemplos.

## Conhecendo as tabelas de sistema

Após finalizarmos a criação do banco de dados, daremos início a uma explicação introdutória sobre cada uma das tabelas do banco e suas respectivas funcionalidades. É importante observamos que através destes recursos podemos realizar qualquer tipo de manipulação ou validação na base de dados, e é claro, possibilitar o desenvolvimento de aplicações que nos permitam ou facilitem essas ações. Observe estas informações na **Tabela 1.**

**Nota**

Devido ao enorme número de tabelas não será possível realizar exemplos com todas as tabelas, porém, iremos abordar as principais, ou mais úteis.

## Aplicando os exemplos no banco de dados

Iniciaremos vendo em que situações podemos utilizar a tabela RDB$DATABASE. Por possuir apenas uma única linha e uma estrutura muito simples, composta por apenas quatro campos, esta tabela costuma ser utilizada em conjunto com alguma função do banco de dados e na realização de tarefas simples. É comum vermos algo do tipo:

```
SELECT GEN_ID(GENERATOR_NAME, 1) FROM RDB$DATABASE
```

Neste caso, a função GEN_ID retorna o próximo valor do GENERATOR especificado baseando-se na quantidade definida no segundo parâmetro, neste caso ele consideraria o valor atual somado a 1. Outro exemplo de utilização é realizado em casos onde necessita-se retornar os formatos de data, hora ou data e hora do servidor através do respectivos métodos CURRENT_DATE, CURRENT_TIME e CURRENT_TIMESTAMP. Podemos utilizar para estes casos as seguintes sentenças:

```
SELECT CURRENT_DATE FROM RDB$DATABASE
SELECT CURRENT_TIME FROM RDB$DATABASE
SELECT CURRENT_TIMESTAMP FROM RDB$DATABASE
```

**Nota**

É possível na criação de domínios definir estes valores como padrão, ou seja, para um campo DATE é possível colocar seu valor DEFAULT como CURRENT_DATE tendo assim seu preenchimento automático no momento da inserção.

## Visualizando as dependências

Através da tabela RDB$DEPENDENCIES é possível realizar o mapeamento das dependências de um objeto do banco de dados. Observe a seguinte instrução:

```
SELECT
  RDB$DEPENDENT_NAME,
  RDB$DEPENDED_ON_NAME,
  RDB$FIELD_NAME
FROM
  RDB$DEPENDENCIES
WHERE
  RDB$DEPENDENCIES.RDB$DEPENDED_ON_NAME = 'PESSOA' AND
  RDB$FIELD_NAME IS NOT NULL
```

Na instrução anterior, realizamos uma consulta na tabela RDB$DEPENDENCIES, obtendo o nome da dependência do objeto (RDB$DEPENDENT_NAME), o objeto dependente em questão (RDB$DEPENDED_ON_NAME) e o nome do campo (RDB$FIELD_NAME) e informando as condições ao filtro, que o objeto dependente em questão é a tabela PESSOA e devem ser desprezados os campos nulos. O resultado obtido é exibido na **Figura 2**.

Observando a **Figura 2** é possível deduzirmos quais são os objetos dependentes da tabela PESSOA. Um exemplo mais claro desta funcionalidade pode ser notado na própria IDE

**Listagem 1.** Script para criação do banco de dados

```
SET SQL DIALECT 3;
SET NAMES WIN1252;

SET CLIENTLIB '<DIRETÓRIO DA INSTALAÇÃO DO FIREBIRD>\BIN\fbclient.
dll';

CREATE DATABASE '<DIRETÓRIO DO BANCO DE DADOS>\CLUBEDELPHI.FDB'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 8192

DEFAULT CHARACTER SET WIN1252;

/*DOMAINS*/
CREATE DOMAIN D_INTEIRO INTEGER;
CREATE DOMAIN D_DESCRICAO VARCHAR(250);
CREATE DOMAIN D_TIPO_PESSOA AS CHAR(1) CHARACTER SET WIN1252
 DEFAULT 'C' CHECK ((VALUE = 'C') OR (VALUE = 'F'));
CREATE DOMAIN D_CEP CHAR(8);
CREATE DOMAIN D_ESTADO CHAR(1);
CREATE DOMAIN D_DATA DATE;

/*GENERATORS*/
CREATE GENERATOR GEN_IDDEPARTAMENTO;
CREATE GENERATOR GEN_IDPESSOA_DEPARTAMENTO;
CREATE GENERATOR GEN_IDENDERECO;
CREATE GENERATOR GEN_IDPESSOA;

/*TABLES*/
CREATE TABLE ENDERECO
(
IDENDERECO D_INTEIRO NOT NULL,
ENDERECO   D_DESCRICAO NOT NULL,
NUMERO     D_INTEIRO NOT NULL,
BAIRRO     D_DESCRICAO NOT NULL,
CIDADE     D_DESCRICAO NOT NULL,
ESTADO     D_ESTADO NOT NULL,
CEP        D_CEP NOT NULL
);

CREATE TABLE PESSOA
(
IDPESSOA        D_INTEIRO NOT NULL,
NOME            D_DESCRICAO NOT NULL,
DATA_NASCIMENTO D_DATA NOT NULL,
DATA_CADASTRO   D_DATA NOT NULL,
TIPO_PESSOA     D_TIPO_PESSOA NOT NULL,
IDENDERECO      D_INTEIRO NOT NULL
);

CREATE TABLE DEPARTAMENTO
(
IDDEPARTAMENTO D_INTEIRO NOT NULL,
DEPARTAMENTO   D_DESCRICAO NOT NULL
);

CREATE TABLE PESSOA_DEPARTAMENTO
(
IDPESSOA_DEPARTAMENTO D_INTEIRO NOT NULL,
IDDEPARTAMENTO        D_INTEIRO NOT NULL,
IDPESSOA              D_INTEIRO NOT NULL
);

/*CONSTRAINTS*/
ALTER TABLE ENDERECO
ADD CONSTRAINT PK_IDENDERECO
PRIMARY KEY(IDENDERECO)
USING INDEX IDX_ENDERECO;

ALTER TABLE PESSOA
ADD CONSTRAINT PK_IDPESSOA
PRIMARY KEY (IDPESSOA)
USING INDEX IDX_IDPESSOA;

ALTER TABLE PESSOA
ADD CONSTRAINT FK_PESSOA_ENDERECO
FOREIGN KEY (IDENDERECO)
REFERENCES ENDERECO (IDENDERECO)
ON UPDATE CASCADE
ON DELETE CASCADE
USING INDEX IDX_PESSOA_ENDERECO;

ALTER TABLE DEPARTAMENTO
ADD CONSTRAINT PK_IDDEPARTAMENTO
PRIMARY KEY (IDDEPARTAMENTO)
USING INDEX IDX_IDDEPARTAMENTO;

ALTER TABLE PESSOA_DEPARTAMENTO
ADD CONSTRAINT PK_PESSOA_DEPARTAMENTO
PRIMARY KEY (IDPESSOA_DEPARTAMENTO)
USING INDEX IDX_PESSOA_DEPARTAMENTO;

ALTER TABLE PESSOA_DEPARTAMENTO
ADD CONSTRAINT FK_DEPARTAMENTO
FOREIGN KEY (IDDEPARTAMENTO)
REFERENCES DEPARTAMENTO (IDDEPARTAMENTO)
ON DELETE CASCADE
ON UPDATE CASCADE
USING INDEX IDX_FK_IDDEPARTAMENTO;

ALTER TABLE PESSOA_DEPARTAMENTO
ADD CONSTRAINT FK_PESSOA
FOREIGN KEY (IDPESSOA)
REFERENCES PESSOA (IDPESSOA)
ON DELETE CASCADE
ON UPDATE CASCADE
USING INDEX IDX_FK_IDPESSOA;

/*TRIGGERS*/
CREATE TRIGGER PESSOA_BI FOR PESSOA
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.IDPESSOA IS NULL) THEN
    NEW.IDPESSOA = GEN_ID(GEN_IDPESSOA,1);
END;

CREATE TRIGGER ENDERECO_BI FOR ENDERECO
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.IDENDERECO IS NULL) THEN
    NEW.IDENDERECO = GEN_ID(GEN_IDENDERECO,1);
END;

CREATE TRIGGER PESSOA_DEPARTAMENTO_BI FOR PESSOA_DEPARTAMENTO
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.IDPESSOA_DEPARTAMENTO IS NULL) THEN
    NEW.IDPESSOA_DEPARTAMENTO = GEN_ID(GEN_IDPESSOA_DEPARTAMENTO,1);
END;

CREATE TRIGGER DEPARTAMENTO_BI FOR DEPARTAMENTO
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.IDDEPARTAMENTO IS NULL) THEN
    NEW.IDDEPARTAMENTO = GEN_ID(GEN_IDDEPARTAMENTO,1);
END;
```

do IBExpert, onde ao selecionarmos a tabela PESSOA e em seguida a aba Dependencies veremos em forma de TreeView o mesmo resultado, como mostrado na **Figura 3**.

## Capturando os objetos de exceção do banco de dados

É comum que ao realizarmos algum tipo de validação no banco de dados, façamos o uso de alguma EXCEPTION para o lançamento de mensagens personalizadas. Existem casos onde

| RDB$DEPENDENT_NAME | RDB$DEPENDED_ON_NAME | RDB$FIELD_NAME |
|---|---|---|
| CHECK_1 | PESSOA | IDENDERECO |
| CHECK_2 | PESSOA | IDENDERECO |
| CHECK_11 | PESSOA | IDPESSOA |
| CHECK_12 | PESSOA | IDPESSOA |
| PESSOA_BI | PESSOA | IDPESSOA |
| V_PESSOA_CLIENTE | PESSOA | TIPO_PESSOA |
| V_PESSOA_CLIENTE | PESSOA | IDENDERECO |
| V_PESSOA_CLIENTE | PESSOA | IDPESSOA |
| V_PESSOA_CLIENTE | PESSOA | NOME |
| V_PESSOA_CLIENTE | PESSOA | DATA_NASCIMENTO |
| V_PESSOA_CLIENTE | PESSOA | DATA_CADASTRO |

**Figura 2.** Resultado da consulta

| RDB$CHARACTER_SETS | Descreve os grupos de caracteres válidos no BD. |
|---|---|
| RDB$COLLATIONS | Lista os COLLATES disponíveis ao usuário no BD. |
| RDB$CHECK_CONSTRAINTS | Armazena os dados de integridade referencial definidos pelas CHECKs e pelo uso da opção NOT NULL. |
| RDB$DATABASE | Define o BD propriamente dito. Essa tabela tem sempre um único registro. |
| RDB$DEPENDENCIES | Armazena as dependências de tabelas e campos com outros objetos do BD como, por exemplo, VIEWS, TABELAS, TRIGGERS e campos calculados. O Firebird utiliza essa tabela para checar se um objeto pode ser removido caso ele não tenha nenhuma dependência |
| RDB$EXCEPTIONS | Define as exceções geradas por erros, incluindo as exceções definidas pelo usuário. |
| RDB$FIELD_DIMENSIONS | Contém as dimensões das colunas do tipo ARRAY. |
| RDB$FIELDS | Define as características de um campo. Todo campo ou domínio tem um registro correspondente nessa tabela. |
| RDB$FILES | Define os arquivos de Shadow do BD bem como os arquivos secundários que podem compor o BD. |
| RDB$FILTERS | Contém informações sobre os filtros do BLOB. |
| RDB$FORMATS | Mantém informações sobre os formatos das colunas das tabelas. Cada vez que o tipo/formato de um campo é alterado, o Firebird atribui um novo número de formato à tabela, permitindo que os aplicativos continuem acessando essa tabela sem que sejam recompilados. |
| RDB$FUNCTION_ARGUMENTS | Define os atributos dos argumentos de uma função. |
| RDB$FUNCTIONS | Armazena a definição das UDFs (Funções Definidas pelo Usuário). |
| RDB$GENERATORS | Armazena as informações sobre os GENERATORs. |
| RDB$INDEX_SEGMENTS | Especifica os campos que compõem um índice de uma tabela. A alteração de registros nessa tabela sem que sejam apagados e recriados os registros na tabela RDB$INDICES dentro da mesma transação pode gerar corrupção no BD. |
| RDB$INDICES | Define as estruturas dos índices que permitem que o IB localize registros de maneira eficiente. Cada índice definido nessa tabela deve ter um registro associado na tabela RDB$INDEX_SEGMENTS. |
| RDB$LOG_FILES | Não é utilizada. |
| RDB$PAGES | Mantém uma relação de páginas alocadas para o BD. É uma tabela que deve ser manipulada com cuidado, pois se for alterada provoca corrupção no BD. |
| RDB$PROCEDURE_PARAMETERS | Armazena informações sobre cada parâmetro definido em cada Stored Procedure do BD. |
| RDB$PROCEDURES | Mantém as informações sobre as Stored Procedures do BD. |
| RDB$REF_CONSTRAINTS | Armazena as informações sobre a integridade referencial no BD. |
| RDB$RELATION_CONSTRAINTS | Armazena as informações sobre integridade referencial das tabelas do BD. |
| RDB$RELATION_FIELDS | Mantém a lista dos campos das tabelas e as informações das características de uma coluna para os domínios. Os nomes dos campos são relacionados através da coluna RDB$FIELD_SOURCE, com um registro na tabela REDB$FIELDS que contém um nome de sistema ("SQL$<n>"). Os dados armazenados incluem informações sobre os tipos das colunas, e podem conter informações sobre os valores default e possibilidade de NULLs. |
| RDB$RELATIONS | Define algumas das características das TABELAS e VIEWS. As outras características são armazenadas na tabela RDB$RELATION_FIELDS. |
| RDB$ROLES | Armazena os ROLES que foram definidos no BD. Bem como o criador de cada ROLE. |
| RDB$SECURITY_CLASSES | Armazena as listas de controle de acesso associando-as com as tabelas, VIEWS, campos e BD. As informações sobre os objetos SQL são duplicadas na tabela RDB$USER_PRIVILEGES. |
| RDB$TRANSACTIONS | Mantém a lista das transações entre múltiplos bancos de dados. |
| RDB$TRIGGER_MESSAGES | Define as mensagens dos TRIGGERS relacionando-as com cada TRIGGER. |
| RDB$TRIGGERS | Define as TRIGGERs. |
| RDB$TYPES | A capacidade dessa tabela não é utilizada na versão atual do Firebird. Ela armazena tipos de dados enumerados e apelidos para charsets e collates. |
| RDB$USER_PRIVILEGES | Mantém as informações sobre os privilégios atribuídos aos usuários através do comando GRANT. |
| RDB$VIEW_RELATIONS | Não é utilizada pelos objetos SQL. |

**Tabela 1.** Tabelas do sistema

os DBAs fazem uso de vários destes objetos relacionando-os suas mensagens com vários objetos distintos (normalmente Triggers), no entanto, é possível ter um único objeto, e fazer o lançamento de uma mensagem personalizada toda vez que houver necessidade. Crie uma EXCEPTION executando o seguinte script:

```
CREATE EXCEPTION E_GERAL '';
```

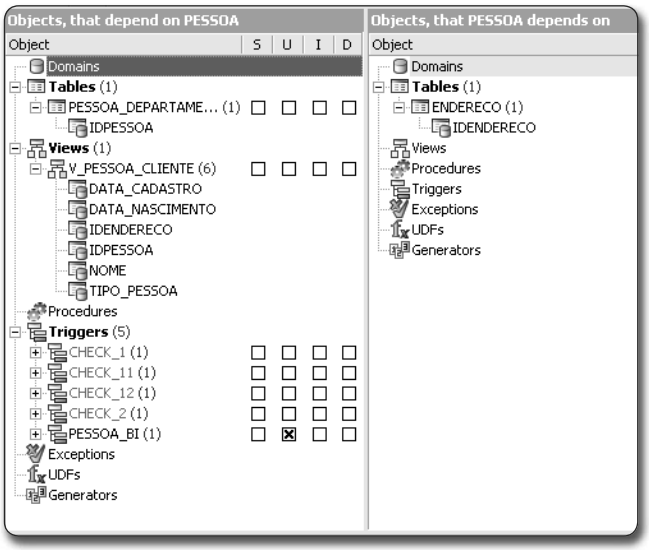



**Figura 3.** Tabela RDB$DEPENDENCIES no IBExpert

No código anterior simplesmente fizemos a criação de uma exception que não possui nenhuma mensagem previamente definida. Utilizando a tabela RDB$EXCEPTIONS é possível obtermos a lista de exceções do banco de dados e realizarmos alterações nas mesmas. Observe a sentença a seguir e veja o resultado na **Figura 4**.

```
SELECT
  RDB$EXCEPTION_NUMBER,
  RDB$EXCEPTION_NAME,
  RDB$MESSAGE
FROM
  RDB$EXCEPTIONS
```

Executando esta SQL veremos que o resultado é mesmo da **Figura 5**.

| RDB$EXCEPTION_NUMBER | RDB$EXCEPTION_NAME | RDB$MESSAGE |
|---|---|---|
| 1 | E_GERAL | |

**Figura 4.** Consulta na tabela RDB$EXCEPTIONS

No IBExpert selecionando as Exceptions e clicando sobre elas no TreeView do lado direito, será exibida uma tela de navegação e manutenção muito parecida com esta, que nos permite o manuseio conforme a necessidade como demonstra a **Figura 5.**

NEW_EXCEPTION ''

| Exception ID | Exception name | Exception text |
|---|---|---|
| 1 | E_GERAL | |

**Figura 5.** Tela de manutenção de exceções do IBExpert

## Obtendo informações dos domínios

Como já demonstrado em meu artigo de personalização de aplicações (ClubeDelphi 112) é possível obtermos informações sobre os nossos domínios. A tabela RDB$FIELDS possui informações relacionadas aos campos e domínios do banco de dados. Com isso torna-se possível o mapeamento dos domínios, possibilitando inclusive saber as definições de valores default ou regras de validação. Atente para a instrução SQL a seguir:

```
SELECT
  RDB$FIELD_NAME,
  RDB$VALIDATION_SOURCE,
  RDB$DEFAULT_SOURCE
FROM
  RDB$FIELDS
WHERE
  RDB$FIELD_NAME NOT STARTING WITH 'RDB$'
```

Na instrução anterior realizamos exatamente o que foi dito, neste caso trazemos a informação do nome do campo (domínio), a regra de validação e o valor padrão (caso exista). O filtro neste caso define que o nome do domínio não deve iniciar em 'RDB$', trazendo assim apenas o que foi definido por nós, como mostrado na **Figura 6.**

| RDB$FIELD_NAME | RDB$VALIDATION_SOURCE | RDB$DEFAULT_SOURCE |
|---|---|---|
| D_INTEIRO | <null> | <null> |
| D_DESCRICAO | <null> | <null> |
| D_TIPO_PESSOA | CHECK ((VALUE = 'C') OR (VALUE = 'F')) | DEFAULT 'C' |
| D_CEP | <null> | <null> |
| D_DATA | <null> | <null> |
| D_ESTADO | <null> | <null> |

**Figura 6.** Domínios definidos por nós

## Mapeando os Generators

A tabela RDB$GENERATORS contém a lista de todos os objetos GENERATOR do banco de dados. É muito comum utilizarmos estratégias de criação de GENERATORS em tempo de execução em nossas aplicações, ou mesmo dentro do banco de dados. Para isso, é ideal que você utilize um padrão de nomenclatura de seus objetos na qual permita uma verificação de existência do objeto que estará sendo criado. Observe a seguinte instrução e seu resultado na **Figura 7**.

```
SELECT
  RDB$GENERATOR_NAME
FROM
  RDB$GENERATORS
```

| RDB$GENERATOR_NAME |
|---|
| RDB$SECURITY_CLASS |
| SQL$DEFAULT |
| RDB$PROCEDURES |
| RDB$EXCEPTIONS |
| RDB$CONSTRAINT_NAME |
| RDB$FIELD_NAME |
| RDB$INDEX_NAME |
| RDB$TRIGGER_NAME |
| RDB$BACKUP_HISTORY |
| GEN_IDDEPARTAMENTO |
| GEN_IDDEPARTAMENTO_PESSOA |
| GEN_IDENDERECO |
| GEN_IDPESSOA |
| GEN_IDPESSOA_DEPARTAMENTO |

**Figura 7.** Resultado da sentença SQL na tabela RDB$GENERATORS sem filtro

Observando o resultado obtido na execução da SQL, veremos que são listados alguns objetos que não foram criados por nós (**Figura 7**). Porém, como utilizamos a nomenclatura GEN_ para a definição dos GENERATORS, podemos realizar um filtro para visualizarmos apenas os objetos por nós definidos, como a sentença a seguir:

```
SELECT
  RDB$GENERATOR_NAME
FROM
  RDB$GENERATORS
WHERE
 RDB$GENERATOR_NAME STARTING WITH 'GEN'´
```

O resultado exibido agora será o da **Figura 8.**

| RDB$GENERATOR_NAME |
|---|
| GEN_IDDEPARTAMENTO |
| GEN_IDDEPARTAMENTO_PESSOA |
| GEN_IDENDERECO |
| GEN_IDPESSOA |
| GEN_IDPESSOA_DEPARTAMENTO |

**Figura 8.** Resultado da sentença SQL na tabela RDB$GENERATORS com filtro

Já que estamos abordando a criação de objetos em tempo de execução ou diretamente pelo banco de dados, iremos criar uma PROCEDURE que receberá o nome de um GENERATOR como parâmetro para checar se o mesmo já existe, para que, caso contrário seja automaticamente criado. Esta PROCEDURE será abordada novamente mais adiante. O código de criação é visto na **Listagem 2**.

**Listagem 2.** Declaração da procedure PROC_VERIFICA_GEN

```
CREATE PROCEDURE PROC_VERIFICA_GEN (P_GEN_NAME VARCHAR(50))
AS
  DECLARE VARIABLE V_COUNT INTEGER;
BEGIN
  SELECT
    COALESCE(COUNT(RDB$GENERATOR_NAME),0)
  FROM
    RDB$GENERATORS
  WHERE
    RDB$GENERATOR_NAME = :P_GEN_NAME
  INTO
    :V_COUNT;

  IF (V_COUNT = 0) THEN
    EXECUTE STATEMENT 'CREATE GENERATOR GEN_'
     || :P_GEN_NAME;
END;
```

Execute em seguida a instrução:

```
EXECUTE PROCEDURE PROC_VERIFICA_GEN('CLUBE_DELPHI');
```

Automaticamente nossa procedure se encarregará de criar um objeto chamado GEN_CLUBE_DELPHI, já que o mesmo não consta no banco de dados. Verifique sua criação executando as instruções anteriores para consulta dos GENERATORS.

## Obtendo informações relacionadas aos índices

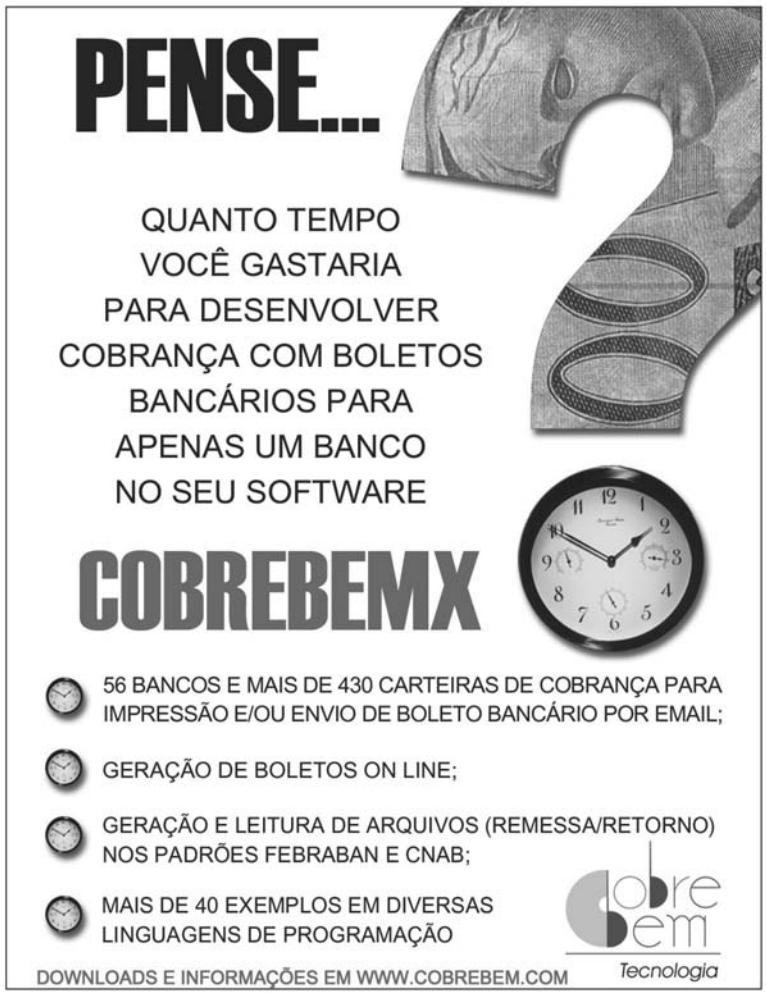
Assim como na tabela RDB$GENERATORS, as tabelas que contém informações sobre os índices (RDB$INDEXES e RDB$INDEX_SEGMENTS) possuem dados relacionados a outros objetos do banco de dados. Como dito anteriormente no começo do artigo, cada registro na tabela RDB$INDEXES possui um registro associado à tabela RDB$INDEX_SEGMENTS, pois a tabela RDB$INDEX_SEGMENTS funciona como uma tabela de integridade dos índices, portanto, para simplesmente mapear os índices não necessitamos desta tabela (a não ser que façamos algo para exibir esta integridade, que não é o nosso caso). Execute a sentença da listagem a seguir para obter o nome dos índices do banco de dados:

```
SELECT
  RDB$INDEX_NAME,
  RDB$RELATION_NAME
FROM
  RDB$INDICES
```

Note na **Figura 9** que a primeira coluna (RDB$INDEX_NAME) exibe o nome dos índices e a segunda mostra o nome das relações. Perceba que como dito anteriormente, esta sentença SQL exibe todos os objetos dentro da RDB$INDEXES.

Mais uma vez é muito simples exibir apenas os campos definidos por nós, bastando apenas acrescentar a seguinte instrução na condição where da SQL anterior:

```
…
WHERE
  RDB$INDEX_NAME NOT STARTING WITH 'RDB'
```



O resultado desta vez exibirá as últimas sete linhas da **Figura 9.** É possível vermos um exemplo deste filtro no próprio TreeView do IBExpert, como mostra a **Figura 10.**

| RDB$INDEX_NAME | RDB$RELATION_NAME |
|---|---|
| RDB$INDEX_34 | RDB$VIEW_RELATIONS |
| RDB$INDEX_35 | RDB$TRIGGER_MESSAGES |
| RDB$INDEX_36 | RDB$FIELD_DIMENSIONS |
| RDB$INDEX_37 | RDB$TYPES |
| RDB$INDEX_38 | RDB$TRIGGERS |
| RDB$INDEX_39 | RDB$ROLES |
| RDB$INDEX_40 | RDB$CHECK_CONSTRAINTS |
| RDB$INDEX_41 | RDB$INDICES |
| RDB$INDEX_42 | RDB$RELATION_CONSTRAINTS |
| RDB$INDEX_43 | RDB$RELATION_CONSTRAINTS |
| RDB$INDEX_44 | RDB$BACKUP_HISTORY |
| RDB$INDEX_45 | RDB$FILTERS |
| IDX_ENDERECO | ENDERECO |
| IDX_IDPESSOA | PESSOA |
| IDX_PESSOA_ENDERECO | PESSOA |
| IDX_IDDEPARTAMENTO | DEPARTAMENTO |
| IDX_PESSOA_DEPARTAMENTO | PESSOA_DEPARTAMENTO |
| IDX_FK_IDDEPARTAMENTO | PESSOA_DEPARTAMENTO |
| IDX_FK_IDPESSOA | PESSOA_DEPARTAMENTO |

**Figura 9.** Resultado da SQL sem filtros

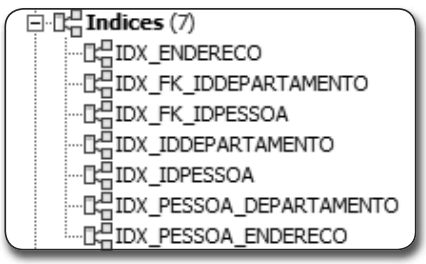



**Figura 10.** TreeView de índices do IBExpert

## Trabalhando com procedures

Para trabalhar com PROCEDURES, basicamente utilizamos duas tabelas temporárias: RDB$PROCEDURE_PARAMETERS e RDB$PROCEDURES. A primeira como o próprio nome diz, é responsável pela listagem das informações sobre os parâmetros de cada uma das PROCEDURES existentes no banco de dados. Veja a instrução a seguir:

```
SELECT
  RDB$PARAMETER_NAME
FROM
  RDB$PROCEDURE_PARAMETERS
WHERE
  RDB$PROCEDURE_NAME = 'PROC_VERIFICA_GEN'
```

Ao criarmos a procedure PROC_VERIFICA_GEN definimos um parâmetro de entrada e nenhum parâmetro de saída, já que nosso objeto apenas executa uma ação de criação de GENERATOR's em tempo de execução. Na instrução anterior, pedimos para que fossem listados os parâmetros de nossa procedure utilizando a tabela RDB$PROCEDURE_PARAMETERS. A seguir, faremos a listagem dos objetos do banco utilizando a tabela RDB$PROCEDURES. Execute a listagem a seguir:

```
SELECT
  RDB$PROCEDURE_NAME,
  RDB$PROCEDURE_INPUTS,
  RDB$PROCEDURE_OUTPUTS,
  RDB$PROCEDURE_SOURCE
FROM
  RDB$PROCEDURES
```

A instrução anterior exibe o nome da procedure no banco de dados, o número de parâmetros de entrada, o número de parâmetros de saída e por último na coluna RDB$PROCEDURE_SOURCE a estrutura da procedure sem o cabeçalho de criação. O resultado você pode observar na **Figura 11.**

| RDB$PROCEDURE_NAME | RDB$PROCEDURE_INPUTS | RDB$PROCEDURE_OUTPUTS | RDB$PROCEDURE_SOURCE |
|---|---|---|---|
| PROC_VERIFICA_GEN | 1 | <null> | DECLARE VARIABLE |

**Figura 11.** Execução da instrução SQL na tabela RDB$PROCEDURES

## Obtendo os campos de cada tabela

Como já mostrado anteriormente no artigo digital da edição 112 da ClubeDelphi, é possível mapearmos os campos de uma tabela, ou view do banco de dados. Naquele caso utilizamos uma estratégia de armazenamento dos FIELDS em uma tabela por nós criada, isso fazia com que tivéssemos nossa estrutura de campos totalmente atualizada e permitia-nos definir uma estratégia de configuração dos TFields dos DataSets. Veremos a seguir um exemplo de sentença SQL na qual realiza a busca na tabela RDB$RELATION_FIELDS:

```
SELECT
  RDB$FIELD_NAME,
  RDB$RELATION_NAME,
  RDB$FIELD_SOURCE
FROM
  RDB$RELATION_FIELDS
WHERE
  RDB$FIELD_NAME NOT STARTING WITH 'RDB' AND
  RDB$BASE_FIELD IS NULL
```

Se notarmos as condições do where, veremos que há uma onde RDB$BASE_FIELD deve ser nulo. Esta condição foi adotada para que listassemos apenas os campos referentes às tabelas do banco de dados e nenhuma estrutura referente às VIEWS. O campo RDB$FIELD_NAME representa o nome dos campos, RDB$RELATION_NAME refere-se ao nome da tabela na qual o campo pertence e por último, RDB$FIELD_SOURCE refere-se ao domínio ou tipo do campo. Execute a sentença anterior e note o resultado como mostrado na **Figura 12.**

| RDB$FIELD_NAME | RDB$RELATION_NAME | RDB$FIELD_SOURCE |
|---|---|---|
| IDENDERECO | ENDERECO | D_INTEIRO |
| ENDERECO | ENDERECO | D_DESCRICAO |
| NUMERO | ENDERECO | D_INTEIRO |
| BAIRRO | ENDERECO | D_DESCRICAO |
| CIDADE | ENDERECO | D_DESCRICAO |
| ESTADO | ENDERECO | D_ESTADO |
| CEP | ENDERECO | D_CEP |
| IDPESSOA | PESSOA | D_INTEIRO |
| NOME | PESSOA | D_DESCRICAO |
| DATA_NASCIMENTO | PESSOA | D_DATA |
| TIPO_PESSOA | PESSOA | D_TIPO_PESSOA |
| IDENDERECO | PESSOA | D_INTEIRO |
| IDDEPARTAMENTO | DEPARTAMENTO | D_INTEIRO |
| DEPARTAMENTO | DEPARTAMENTO | D_DESCRICAO |
| IDPESSOA_DEPARTAMENTO | PESSOA_DEPARTAMENTO | D_INTEIRO |
| IDDEPARTAMENTO | PESSOA_DEPARTAMENTO | D_INTEIRO |
| IDPESSOA | PESSOA_DEPARTAMENTO | D_INTEIRO |
| DATA_CADASTRO | PESSOA | D_DATA |

**Figura 12.** Execução da instrução SQL para exibição dos campos

| | RDB$TRIGGER_NAME | RDB$RELATION_NAME | RDB$TRIGGER_SOURCE |
|---|---|---|---|
| | DEPARTAMENTO_BI | DEPARTAMENTO | AS |
| | ENDERECO_BI | ENDERECO | AS |
| | PESSOA_BI | PESSOA | AS |
| | PESSOA_DEPARTAMENTO_BI | PESSOA_DEPARTAMENTO | AS<br>BEGIN<br>IF (NEW.IDPESSOA_DEPARTAMENTO IS NULL) THEN<br>NEW.IDPESSOA_DEPARTAMENTO = GEN_ID(GEN_IDPESSOA_DEPARTAMENTO,1);<br>END |
| * | <null> | <null> | |

**Figura 13.** Execução da instrução SQL - Triggers

## Mapeando Triggers

Por último, iremos visualizar como mapear as Triggers do banco de dados. Este processo é muito parecido com o que foi feito na tabela RDB$PROCEDURES, pois esta também possui informações sobre as validações do banco de dados, ou seja, Triggers não definidas por nós. Para listarmos apenas os objetos criados por nós simplesmente devemos validar a coluna RDB$TRIGGER_SOURCE, pois apenas nossos objetos possuirão implementação. A sentença que deve ser executada desta vez é a da listagem a seguir:

```
SELECT
  RDB$TRIGGER_NAME,
  RDB$RELATION_NAME,
  RDB$TRIGGER_SOURCE
FROM
  RDB$TRIGGERS
WHERE
  RDB$TRIGGER_SOURCE IS NOT NULL
```

A primeira coluna (RDB$TRIGGER_NAME) exibe o nome da Trigger, RDB$RELATION_NAME define a qual tabela pertence aquele determinado objeto e por último a coluna RDB$TRIGGER_SOURCE, que é responsável pela exibição da estrutura ou implementação do objeto. O resultado desta SQL é mostrado na **Figura 13**.

## Conclusão

Com isso finalizamos nosso artigo, como foi visto é possível realizar o mapeamento de praticamente todos os objetos do banco de dados. Apesar de conhecermos apenas uma parte das tabelas e de maneira simplificada, é possível termos uma ideia do que podemos fazer com este conhecimento das tabelas de sistema, permitindo aplicá-los em seus próprios bancos de dados através de estratégias de performance e/ou manutenção ou mesmo desenvolver seus próprios softwares de manutenção a banco de dados, como o caso da ferramenta IBExpert que implementa muitos dos recursos vistos durante o artigo. Um grande abraço e até a próxima! ●

**Links**

**Firebird – Site oficial**
*http://www.firebirdsql.org*

**Downloads do Firebird**
*http://www.firebirdsql.org/index.php?op=files&id=engine*

**Blog do autor**
*http://www.twitter.com/daniel_laporte*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

**Nesta seção você encontra artigos sobre técnicas que poderão aumentar a qualidade do desenvolvimento de software**

# Persistência de Objetos

## Usando o Delphi com o mais popular e poderoso framework de Mapeamento Objeto / Relacional

**Do que trata o artigo**

A persistência de dados toma grande parte do tempo de desenvolvimento de nossos sistemas. Neste artigo veremos como automatizar essas tarefas repetitivas de forma elegante, usando uma ferramenta OR/M (mapeamento objeto relacional). Com seu uso poderemos utilizar objetos de negócio em nossos sistemas e não nos preocuparmos como recuperá-los ou mandá-los para um banco relacional. Vamos ver a teoria por trás de uma ferramenta de mapeamento objeto/relacional, como o NHibernate, e como isso pode melhorar nosso desenvolvimento diário.

**Para que serve**

Uma ferramenta de persistência tem a tarefa de saber como salvar seus objetos em memória, para um dispositivo físico. Isso é feito através do mapeamento objeto relacional, onde apontamos qual campo será utilizado para armazenar determinada propriedade e qual classe/classes representam uma ou mais tabelas.

**Em que situação o tema é útil**

Quando se deseja utilizar a orientação a objetos para representar seu negócio, contudo, sem perder a facilidade de acesso a banco de dados. Quando se deseja ter uma aplicação realmente independente do banco de dados que é utilizado, sem se preocupar com detalhes específicos de cada um.

**Paulo Quicoli**
*pauloquicoli@gmail.com*
*Editor técnico das revistas ClubeDelphi, .Net Magazine e WebMobile. Formado em processamento de dados pela FATEC-TQ. Atua como analista e desenvolvedor da ControlM Informática (www.controlm.com.br) e professor de linguagem de programação do curso técnico em processamento de dados do SENAC/Jaboticabal. Utiliza Delphi desde sua primeira versão, entusiasta do desenvolvimento orientado a objetos, tem publicado vários artigos sobre o assunto. Blog: www.pauloquicoli.spaces.live.com*

### Resumo do DevMan

O NHibernate é uma ferramenta de mapeamento objeto relacional que implementa teorias conceituadas sobre o assunto. Ele vem trazer uma ponte de comunicação entre seus objetos de negócio e seu banco de dados, que podem ser bem diferentes. Isso traz vários benefícios, mas também dificuldades. Um dos benefícios é que suas regras de negócio serão independentes de banco de dados. Para mostrar tudo isso, vamos desenvolver uma aplicação desktop onde nenhum SQL será utilizado, em momento algum. Vamos utilizar um banco de dados Firebird para salvar os objetos criados.

Você, desenvolvedor Delphi, como a grande maioria de usuários, certamente desenvolve aplicações do tipo desktop, em duas ou três camadas. São aplicações centradas em dados, como são conhecidas. É muito comum nesses cenários fazermos a análise do negócio envolvido e já partirmos para o desenvolvimento do banco de dados, onde posteriormente será acessado por controles Data-aware e seus dados manipulados por sentenças SQL geradas por nós mesmos. Ou seja, além de nos preocupar com as regras envolvidas, temos também a responsabilidade de acessar o banco de dados e utilizá-lo. Quero deixar claro que não há nada de errado nisso. Isso tem funcionado muito bem até hoje, contudo, novas técnicas surgiram. Técnicas essas que proporcionam maior flexibilidade, agilidade ao desenvolvimento e diminuem tarefas repetitivas, retirando dos desenvolvedores a preocupação da persistência dos dados. Uma dessas técnicas é a orientação a objetos. Muito se fala nela no mundo Delphi, mas pouco se aplica de verdade. Para mostrar como utilizá-la faremos uma pequena agenda de contatos utilizando o Delphi Prism como linguagem de desenvolvimento, a plataforma .NET e NHibernate como ferramenta objeto relacional. Quanto ao banco de dados, não interessa porque o NHibernate é compatível com cerca de mais de dez produtos existentes no mercado. Vamos começar pelos conceitos que servem de apoio e alicerce.

## Persistência de dados

A persistência, de uma forma geral, é a tarefa de pegar informações contidas em um meio volátil e passá-las para um meio não volátil. Sendo mais específico, é pegar os dados que estão na memória e mandá-los para um HD. Ao se falar de sistemas comerciais, esses dados serão salvos em um banco de dados relacional. Nessa tarefa de persistência temos sub-rotinas, operações, que acontecem o tempo todo durante a execução de um sistema. Esse é o CRUD. Operações de acesso a dados que cuidam da criação, recuperação, atualização e exclusão das informações contidas em um banco de dados.

## Impedância objeto/relacional

Esse é o nome que se dá para as diferenças existentes entre seu banco de dados e seus objetos de negócio. Para exemplificar, imagine que no seu modelo classes temos Pessoa e Endereço. Classes distintas, mas possuem um relacionamento entre si. Pessoa possui em sua interface uma referência à classe Endereço, representando que uma pessoa tem um endereço. Contudo em seu banco de dados você pode ter apenas uma tabela para armazenar as duas informações.

## Mapeamento objeto/relacional

É a operação que visa diminuir a impedância, as diferenças entre o objeto e o relacional. É o local onde definimos como os objetos serão salvos e recuperados do banco de dados. É onde informamos qual classe se torna uma tabela, qual propriedade se torna um campo. Toda ferramenta que se intitula objeto/relacional (ORM) faz de alguma forma esse mapeamento e oferece meios para pesquisa dos objetos salvos. Existem pequenas definições que se aplicam quando vamos fazer um mapeamento objeto relacional. Veremos com mais detalhes durante o desenvolvimento do projeto.

## Mãos à obra

Como já mencionado, vamos utilizar o Framework .NET, Delphi Prism, NHibernate e Firebird para o desenvolvimento de nossa agenda. Nossa agenda irá permitir a inclusão de pessoas, seu endereço, email e telefones. Deverá também oferecer uma forma rápida para localização das pessoas cadastradas. Para começar vamos abrir o Visual Studio 2008 e então acessamos o menu File > New > Project. Configure o projeto como mostra a **Figura 1**.

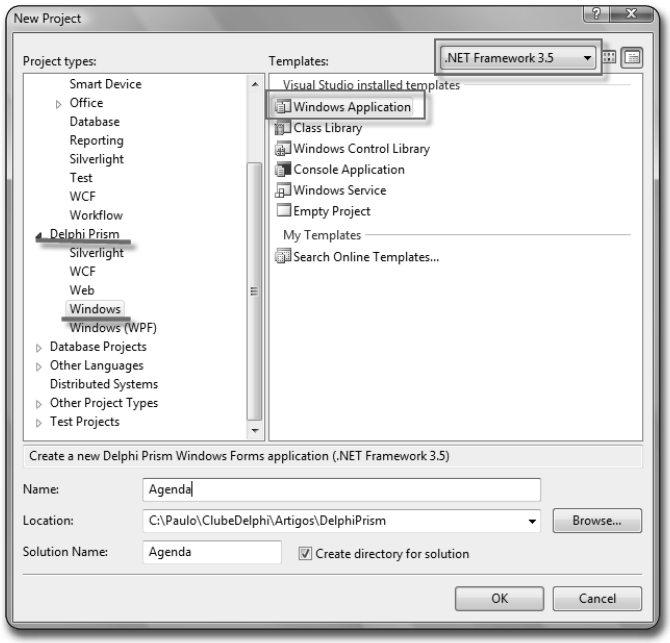



**Figura 1**. Criando um projeto Delphi Prism

Observe que selecionamos a opção Windows de Delphi Prism. Desta forma vamos criar uma aplicação utilizando Windows Forms, que é bem semelhante aos formulários do Delphi win32. Observe também que escolhemos o Framework .net na versão 3.5, o que nos garante utilizar os recursos mais atualizados do mesmo.

### Nota do DevMan

O .NET Framework foi desenvolvido de tal forma que possamos utilizar várias linguagens para o desenvolvimento de sistemas. Existem várias linguagens compatíveis, por exemplo, temos Visual Basic, uma versão do Java conhecida como J#, temos C#, F#, Cobol, o próprio Delphi e muito mais. No caso específico do Delphi, ele funciona como um Plugin dentro do Visual Studio, que é o IDE de desenvolvimento fornecido pela Microsoft. É uma versão evoluída da linguagem Delphi que conhecemos. Nela foram implementados recursos avançados oferecidos pelo .NET Framework. Veja onde baixar o Delphi Prism na seção links.

Ao clicarmos em OK temos um projeto com um formulário principal já criado. Como vamos utilizar a orientação a objetos vamos primeiramente desenvolver as classes de negócio envolvidas. Vamos centralizá-las em um projeto do tipo Class Library. Um projeto desse tipo nada mais é do que uma DLL que pode ser reutilizada em vários projetos. Para criá-la clicamos com o botão direito no nome da solução (Agenda) e no menu que surge, escolhemos Add>New Project. Então, escolhemos Class Library, como mostra a **Figura 2**. Nesse momento seu Solution Explorer deve se parecer com a **Figura 3**.

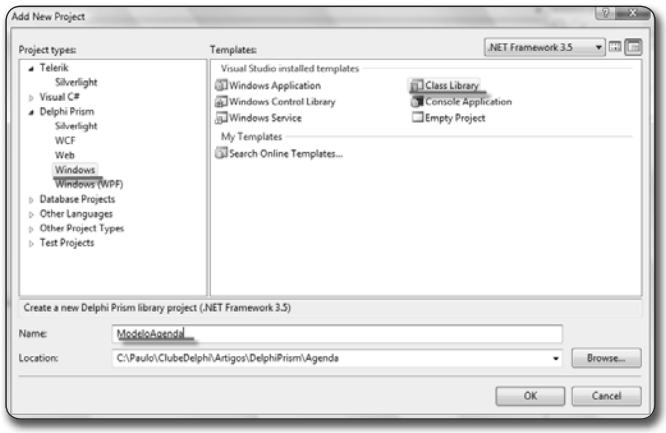



**Figura 2**. Adicionando um novo projeto à solução

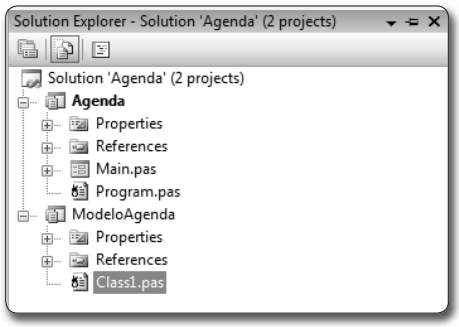



**Figura 3**. Solution Explorer

Veja que temos um arquivo chamado Class1.pas. Clique com o botão direito nele, no menu que aparece escolha Rename e mude seu nome para Pessoa.pas. Altere o nome da classe também no código, para que fique como mostra a **Listagem 1**.

Em nosso sistema, uma pessoa terá um Nome, um Endereco, vários Telefones e um Email. Sendo assim vamos alterar a classe pessoa para que reflita essa definição e fique como mostra a **Listagem 2**.

Veja que agora definimos as propriedades mencionadas. O detalhe fica para a propriedade Endereco, que é do tipo Endereco, uma outra classe que ainda não foi implementada. E também para Telefones, que é uma lista genérica do tipo Telefone, que também ainda não foi implementada. Vamos fazê-lo agora. Clique com o botão direito sobre o projeto ModeloAgenda e escolhe Add>New item. Nessa janela que abrir vamos escolher Class, dando o nome de Telefone. A classe estará vazia. Contudo vamos alterá-la para que fique como na **Listagem 3**.

**Listagem 1**. Classe Pessoa

```
namespace ModeloAgenda;

interface

type
  Pessoa = public class
  private
  protected
  public
  end;

implementation

end.
```

**Listagem 2**. Classe Pessoa

```
type
  Pessoa = public class
  private
  protected
  public
      property Nome: String;
      property Endereco: Endereco;
      property Telefones: IList<Telefone>;
      property Email: String;
      constructor;  end;

implementation

constructor Pessoa;
begin
  Endereco := new Endereco;
  Telefones := new List<Telefone>;
end;
```

**Listagem 3**. Classe Telefone

```
type
  Telefone = public class
  private
  protected
  public
      property Numero: String;
      property Ramal: String;
  end;
```

Até o momento então temos as classes Pessoa e Telefone. Veja que definimos telefone com duas propriedades do tipo string, Numero e Ramal. Falta fazermos agora a classe Endereco. Adicione uma nova classe, lhe dê o nome de Endereco e a codifique como na **Listagem 4**.

**Listagem 4**. Classe Endereco

```
  Endereco = public class
  private
  protected
  public
      property Rua: String;
      property Bairro: String;
      property Cidade: String;
      property UF: String;
      constructor ();
  end;

implementation

constructor Endereco();
begin
  Cidade := "Jaboticabal";
  UF := "SP" ;
end;
```

Vejam que dessa vez temos algo novo. Além das propriedades esperadas para um endereço, incluímos um construtor para a classe. Fazemos isso porque quero que, sempre que um endereço for criado, a Cidade seja "Jaboticabal" e a UF "SP".

Quero que vocês percebam que ainda não mencionamos banco de dados. Da maneira que está já podemos realizar um

protótipo de como será o sistema. Vamos voltar ao formulário principal e vamos deixá-lo como mostra a **Figura 4**.

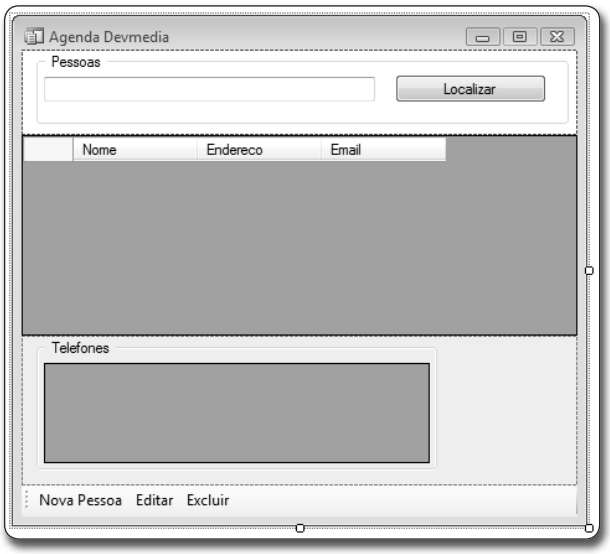



**Figura 4**. Tela principal

Na parte superior do formulário principal temos um Panel com sua propriedade Dock configurada para Top. Dentro dele temos um GroupBox e nele um TextBox e um Button. Depois, na parte inferior, temos um ToolStrip com seu Dock configurado para Bottom. Sobre ele temos também outro Panel, com Dock para Bottom, que possui em seu interior um DataGridView. No centro temos outro DataGridView, porém o Dock deste está configurado para Fill, que preenche toda área livre do formulário. Com essa tela pronta, já é possível realizar uma prototipação, utilizando apenas objetos em memória. É isso que faremos agora.

**Nota**

Ao inserir o DataGridView, não aparecerão as colunas, como mostra a **Figura 4**. Logo veremos como inseri-las.

## Um protótipo

A razão de termos um protótipo é poder mostrar para um possível cliente um sistema funcionando, para que ele possa já, dar um retorno sinalizando se estamos no caminho certo ou não. Então vamos criar um protótipo.

A primeira coisa a se fazer é adicionar uma referência do projeto ModeloAgenda em Agenda. Para isso clique com o botão direito do mouse sobre o projeto Agenda e escolha Add Reference. Na janela que surge escolha a aba Projects e nela aparece ModeloAgenda. Então o selecione e clique em OK. E na seção uses, adicione ModeloAgenda. Agora é possível acessar as classes que definimos no projeto que representa nosso domínio.

A seguir, é preciso criar algumas pessoas para que possam ser listadas no DataGridView. Vamos armazená-las de forma temporária em uma lista de objetos e criar um método que, no construtor do formulário principal, crie 100 pessoas com telefones para nós. Na seção private da classe MainForm (formulário principal) declaramos um campo privado, como a seguir:

```
ListaPessoas: BindingList<Pessoa>;
```

Também definimos logo abaixo dele o método que irá gerar as pessoas:

```
method GerarPessoas:List<Pessoa>;
```

Observe que o método retorna uma lista genérica da classe Pessoa. Na **Listagem 5** vemos sua implementação.

**Listagem 5**. Gerando pessoas

```
method MainForm.GerarPessoas: List<Pessoa>;
var
    pessoas: List<Pessoa>;
    pessoa: Pessoa;
    telefone: Telefone;
    i: Int32;
begin
  pessoas:= new List<Pessoa>;
  for i:= 0 to 100 do
  begin
    pessoa := new Pessoa;
    pessoa.Nome := 'Pessoa '+ i.ToString();
    pessoa.Endereco.Rua := 'Rua número ' + i.ToString();
    pessoa.Endereco.Bairro := 'Centro';
    pessoa.Endereco.Cidade := 'Jaboticabal';
    pessoa.Endereco.UF := 'SP';
    telefone := new Telefone;
    telefone.Numero := '9999-9999';
    telefone.Ramal := i.ToString;
    pessoa.Telefones.Add(telefone);
    pessoas.Add(pessoa);
  end;
  result := pessoas;
end;
```

Este é um método bem simples. Instanciamos uma lista genérica da classe Pessoa em um laço for de 101 repetições, a cada repetição instanciamos uma nova pessoa, um novo telefone e adicionamos a pessoa à lista genérica. Ao término do laço, retornamos a lista gerada.

Isso já nos torna hábeis de ligar ao DataGridView essa lista de objetos. Mas, vamos fazer algo melhor, vamos adicionar um DataSource de objetos ao projeto para que possamos reutilizá-lo. Para criar um novo DataSource, vamos ao menu Data > Add New Data Source. Na janela que surge, escolhemos Object e avançamos. Como mostra a **Figura 5** podemos escolher qual classe vamos acessar.

Nesse momento, escolhemos a classe Pessoa. Avançamos e confirmamos nossa escolha. Ao terminar, será adicionado ao projeto um novo item, como mostra **Figura 6**.

Agora é possível ligar esse DataSource ao DataGridView do formulário principal. Selecione o DataGridView e observe que ao fazer isso, um pequeno botão surge em seu canto superior direito. Esse é o botão Tasks, que centraliza as tarefas mais comuns a se fazer com o objeto selecionado, em nosso caso o DataGridView. Clique nele e abra a lista Choose DataSource, escolhendo um como mostra a **Figura 7**.

Após escolhido, configuramos algumas opções para que o usuário final não consiga alterar o DataGridView (**Figura 8**).

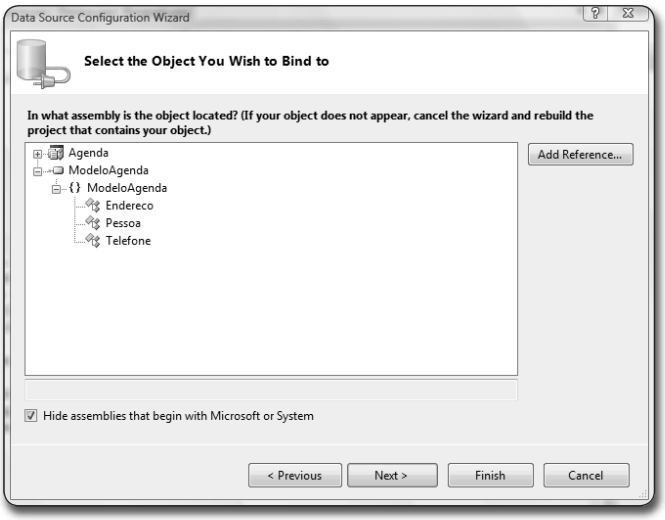



**Figura 5**. Selecionando o objeto para o Data Source

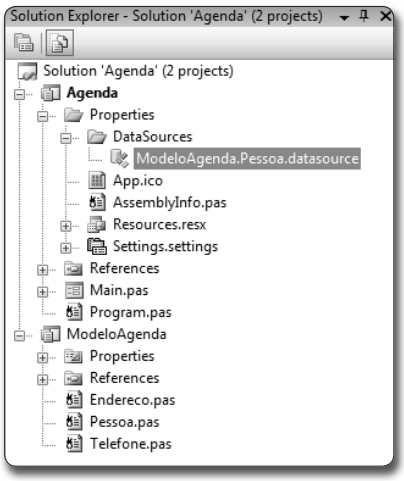



**Figura 6**. DataSource adicionado ao projeto

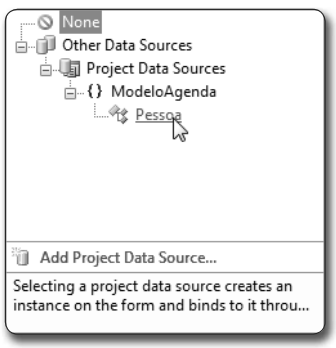



**Figura 7.** Escolhendo o DataSource

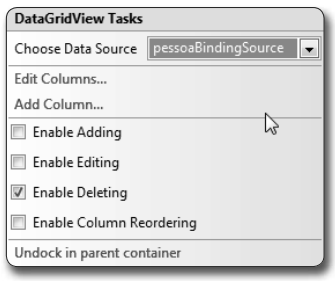



**Figura 8**. Configurando o DataGridView

Ao observar a **Figura 8** veja que temos uma opção Edit Columns, clique nela e configure para que fique como na **Figura 4**. Observe também que ao realizar a ligação (DataBinding) entre o DataGridView e nosso DataSource, foi adicionado ao formulário um controle chamado pessoaBindingSource. Esse controle será responsável pela ligação entre o DataGridView e nosso DataSource. Ele é responsável em manter o posicionamento do elemento selecionado. Agora no construtor do formulário principal podemos criar a lista de pessoas e ligá-la ao BindingSource,veja **Listagem 6**.

**Listagem 6**. Criando a lista de pessoas

```
constructor MainForm;
begin
  //
  // Required for Windows Form Designer support
  //
  InitializeComponent();
  ListaPessoas := new BindingList<Pessoa>(GerarPessoas);
  pessoaBindingSource.DataSource := ListaPessoas;
end;
```

Veja que estamos instanciando o campo privado ListaPessoas. Como ele é do tipo BindingList, seu construtor pode receber uma lista de objetos do tipo especificado. Essa lista em nosso caso, é retornada pelo método GerarPessoas, como mostrado anteriormente. Na sequência ligamos nossa lista à propriedade DataSource de pessoaBindingSource. Como pessoaBindingSource está ligado ao DataGridView, ao fazermos isso, o DataGridView passa a exibir a lista. Execute e veja.

Vamos melhorar a apresentação dos dados, para que, ao escolher uma pessoa, seus e-mails já apareçam no DataGridView inferior. Implementar isso é bem simples. O controle pessoaDataGridView possui um evento chamado CurrentChanged. Esse evento é disparado sempre que o indicador de objeto corrente é movimentado. Então pra resumir, esse evento é disparado sempre que uma posição no DataGridView é movimentada. Implemente o evento como mostra a **Listagem 7**.

**Listagem 7**. Implementando o evento CurrentChanged

```
method MainForm.pessoaBindingSource_CurrentChanged(sender:
  System.Object; e: System.EventArgs);
begin
  gridTelefones.DataSource := (pessoaBindingSource.Current
   as Pessoa).Telefones;
end;
```

Na única linha desse evento estamos obtendo o objeto em foco (selecionado), para então realizar um cast para o tipo Pessoa e então passar para o DataGridView de telefones (gridTelefones) a lista de telefones que a pessoa em foco possui. Assim, ao selecionar uma pessoa, embaixo veremos seu telefone. Execute e veja o programa rodando. Agora vamos implementar o botão excluir. Dê um clique sobre ele e faça como mostra a **Listagem 8**.

Estamos solicitando uma confirmação ao usuário, caso ele confirme, executamos o método RemoveCurrent() de pessoaBindingSource, que remove de sua lista o objeto selecionado. Está

faltando ainda fazer a consulta dessa lista de pessoas. No evento Click do botão localizar insira o código da **Listagem 9**.

**Listagem 8**. Excluindo o objeto selecionado da Lista

```
method MainForm.btnExcluir_Click(sender: System.Object; e:
 System.EventArgs);
begin
    if MessageBox.Show('Confirma exclusão da Pessoa selecionada ?','
     Atenção',MessageBoxButtons.YesNo)=DialogResult.Yes then
    begin
        pessoaBindingSource.RemoveCurrent();
    end;
end;
```

**Listagem 9**.Consultando objetos

```
method MainForm.btnLocalizar_Click(sender: System.Object; e:
 System.EventArgs);
begin
  var resultados := from p in ListaPessoas where p.Nome.
  Contains(txtBusca.Text) select p;
  pessoaBindingSource.DataSource := resultados;
end;
```

Para realizar uma consulta no objeto ListaPessoas, que mantém as pessoas que aparecem no DataGridView, utilizamos uma sentença LINQ (veja nota do DevMan). Então o resultado da consulta é guardado na variável resultados que na sequência é o novo DataSource de pessoaBindingSource. Essa mudança de objetos em pessoaBindingSource.DataSource faz com que o DataGridView seja atualizado, exibindo assim os resultados da consulta. Nosso próximo passo será inclusão e edição.

Adicione um novo formulário ao projeto, clicando com o botão direito sobre o projeto e escolha Add>New Item. Na janela que abre escolha Windows Form e salve como EditPessoa.pas. Ajuste o formulário para que fique como na **Figura 9**.

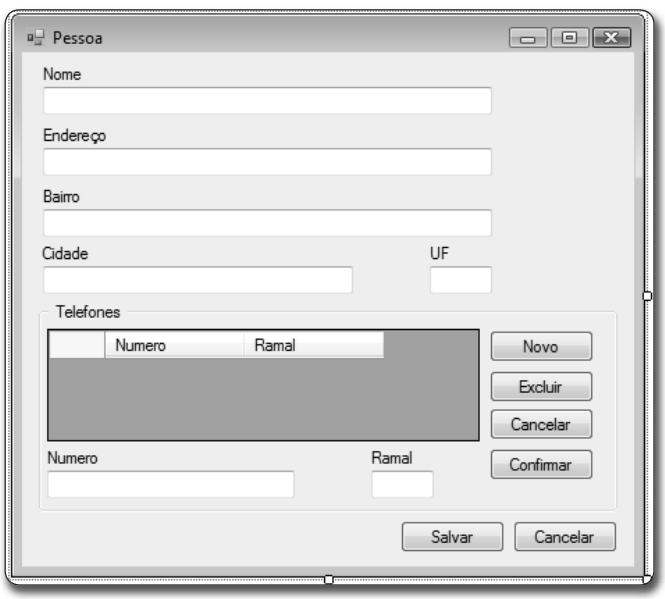



**Figura 9**. Formulário de cadastro de pessoas

Nesse formulário vamos adicionar três controles BindingSource. Um para pessoa, outro para seu endereço e outro para a lista de telefones que uma pessoa pode possuir. Vamos criar o BindingSource para Telefone, que ainda não existe. Acesse o menu Data>Add New DataSource. Escolha Object, avance e escolha Telefone, dentro de ModeloAgenda, depois faça o mesmo para Endereco. Com os DataSources criados, podemos realizar o DataBinding, a ligação dos controles com os dados. Para ligar o TextBox do nome à propriedade Nome de nossa classe pessoa, o selecionamos e acessamos suas propriedades. Nelas teremos um grupo chamado (DataBindings), o expanda e selecione como mostra a **Figura 10**.

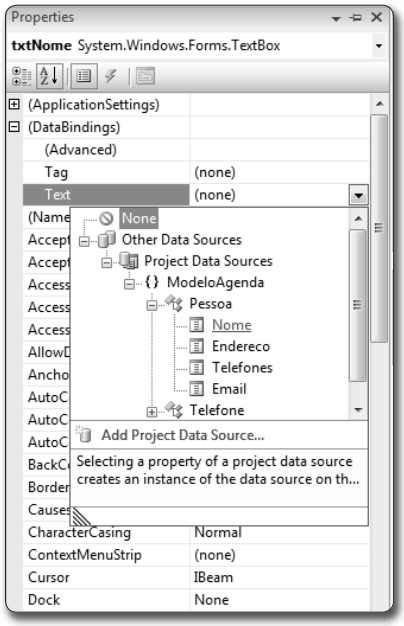



**Figura 10**. Databinding do TextBox Nome

Agora, faça o mesmo para os outros controles, ligando-os aos seus BindingSources correspondentes.

## Editando e inserindo um novo objeto

Como temos tudo ligado, já é possível fazer a edição de um objeto. Inicialmente temos o formulário principal que possui uma consulta de pessoas. A ideia é consultar uma pessoa, selecioná-la e transferir esse objeto para a tela de edição. Para que o formulário de edição receba essa pessoa que vamos transferir, vamos inicialmente criar uma propriedade na seção public da classe do formulário EditPessoa, como a seguir:

```
property PessoaSelecionada: Pessoa;
```

### Nota do DevMan

**LINQ**

De uma forma bem simples podemos dizer que o LINQ é uma nova linguagem que oferece suporte à consulta de dados, que podem estar armazenados nas mais diversas formas possíveis. Os dados a serem consultados podem estar em XML (LINQ para XML), banco de dados diversos e objetos (LINQ to Objects). Ou seja, podemos realizar consultas a essas e outras fontes de dados, através de uma linguagem fortemente tipada.

**Nota**

Adicione na seção uses uma referência ao namespace ModeloAgenda.

Agora no evento Load do formulário ligamos esse objeto PessoaSelecionada a cada BindingSource, **Listagem 10**.

**Listagem 10**. Preenchendo os controles BindingSource

```
method EditPessoa.EditPessoa_Load(sender: System.Object; e:
System.EventArgs);
begin
    pessoaBindingSource.DataSource := PessoaSelecionada;
    enderecoBindingSource.DataSource := PessoaSelecionada.
    Endereco;
    telefoneBindingSource.DataSource := PessoaSelecionada.
    Telefones;
end;
```

Agora vamos voltar ao formulário principal e nele vamos chamar o formulário de edição, no Click do botão editar, **Listagem 11**.

**Listagem 11**. Chamando o formulário de edição

```
method MainForm.btnEditar_Click(sender: System.Object; e:
 System.EventArgs);
var
    form: EditPessoa;
begin
    form := new EditPessoa();
    form.PessoaSelecionada := (pessoaBindingSource.Current as
     Pessoa);
    form.ShowDialog;
end;
```

Ao instanciar o formulário EditPessoa se tornou possível acessar sua propriedade PessoaSelecionada e assim, passar o objeto selecionado no DataGridView. Esse objeto é representado pela propriedade Current de pessoaBindingSource. Execute a aplicação e veja que agora a pessoa selecionada é enviada para a tela de edição.

## Controlando a edição de um objeto

Você pode ter observado que se alterarmos alguma informação do objeto na tela de edição, essa alteração é repassada ao objeto da lista no formulário principal. Também veja que é impossível desfazer essa alteração. Contudo, para ajustar esse comportamento e tornar nossos objetos "editáveis" o Framework .NET disponibiliza uma interface chamada IEditableObject contida no namespace System.ComponentModel. Essa interface basicamente disponibiliza três métodos: BeginEdit, EndEdit e CancelEdit. A ideia sugerida de uso é: antes de realizar alguma alteração no objeto que implementa IEditableObject, chamamos BeginEdit. Para finalizar as alterações, EndEdit e para cancelá-las antes de confirmar, CancelEdit.

Mas essa interface por si só não faz nada, afinal de contas, ela apenas define alguns métodos que são utilizados internamente pelo .NET, em especial pela classe BindingSource. Para que o efeito de desfazer possa ser aplicado, é necessário que no método BeginEdit nós copiemos o conteúdo de cada propriedade para um cache, para que então no método CancelEdit retiremos desse cache os valores originais e os devolvamos às propriedades. E já no método EndEdit, apenas limpamos o cache.

Para aplicar esse comportamento em nossas classes, de forma automática, vamos criar uma nova classe, que servirá de base para nossas classes de negócio. Ela se chamará Entidade. Adicione uma nova classe ao projeto ModeloAgenda, clicando com o botão direito nele e escolhendo Add>New Item>Class. Na nova classe, adicione uma referência à System.ComponentModel e inclua a nova classe Entidade como exibido a seguir, na **Listagem 12**.

**Listagem 12**. Classe Entidade

```
type
  Entidade = public class(IEditableObject)
  private
     props: Hashtable := nil;
  protected
  public
     method EndEdit;
     method CancelEdit;
     method BeginEdit;
  end;

implementation

method Entidade.BeginEdit;
var
    value: Object;
    x: Int32;
begin
     var properties := (self.GetType()).GetProperties
     (BindingFlags.Public + BindingFlags.Instance);

    props := new Hashtable();

    for x:=0 to properties.length-1 do
    begin
        if (properties[x].GetSetMethod()<>nil) then
        begin
            value := properties[x].GetValue(self, nil);
            props.Add(properties[x].Name, value);
        end;
    end;
end;

method Entidade.CancelEdit;
var
    value: Object;
    x: Int32;
begin
   if (props = nil) then exit;

    var properties := (self.GetType()).GetProperties
                (BindingFlags.Public + BindingFlags.Instance);

    for x:=0 to properties.length-1 do
    begin
        if (properties[x].GetSetMethod()<>nil) then
        begin
            value := props[properties[x].Name];
            properties[x].SetValue(self, value, nil);
        end;
    end;
    props := nil;
end;

method Entidade.EndEdit;
begin
    props := nil;
end;
```

Vamos entender o que esse código todo faz. Veja que nossa classe Entidade herda, ou implementa, a Interface IEditableObject. Isso nos obriga a implementar os métodos definidos por ela, são eles: BeginEdit, CancelEdit e EndEdit, que estão declarados na seção public de nossa classe. Para implementar o cache anteriormente citado, vamos fazer uso de um tipo especial do .Net, o HashTable. Ele está em uso na variável props.

O tipo HashTable é como se fosse um vetor bidimensional disposto no formato Chave-Valor. Semelhante em estrutura a um arquivo INI (sem as seções) ou ao registro do Windows, só que em memória. E vamos usá-lo para armazenar os valores originais das propriedades. Para adicionar itens em seu conteúdo, utilizamos seu método Add que requer dois parâmetros: o primeiro é a chave (Key), e o segundo o valor (Value) dessa chave. Onde a chave é o nome da propriedade e o valor é o valor dela mesma. Porém nossa classe base Entidade não possui propriedade alguma, quem possuirá serão seus descendentes. Portanto como obter as propriedades e seus valores? Para isso vamos fazer uso da reflexão.

Observe o método BeginEdit. A primeira linha após o Begin é onde recuperamos, em tempo de execução, a lista de propriedades da classe. Em self temos a instância do objeto em execução. Self, através da inclusão do namespace System.Reflection à seção uses, disponibiliza um método chamado GetType, que retorna o tipo da instância de Self. Sendo assim, se no momento da execução Self for uma instância de Pessoa, o tipo retornado por GetType será ModeloAgenda.Pessoa. E por sua vez, GetType oferece o método GetProperties, que retorna as propriedades da instância (array do tipo PropertyInfo), conforme especificamos nos parâmetros do método. No caso estamos recuperando as propriedades públicas e de instância.

**Nota**

Se você for um bom observador vai notar que a variável properties não foi declarada em lugar algum nem muito menos seu tipo. Muito bem. Esse é um dos recursos do Delphi Prism. Ele permite que variáveis sejam declaradas de forma in-line, ou seja, na própria linha onde será usada, sem a necessidade de se declarar na seção var. Isso traz um grande benefício, a agilidade. Se você tem um código muito grande, porque ter que voltar ao início dele apenas para declarar uma variável? Quanto à outra observação, note a palavra reservada var, antes do nome da variável properties. Esse é um recurso do .Net. Ele faz com que a variável assuma o tipo que lhe é passado no momento da sua criação, e esse tipo não poderá ser alterado posteriormente. Veja, isso é economia de digitação! Mantemos a qualidade de um código fortemente tipado e diminuímos código.

Na linha seguinte instanciamos nosso HashTable para que então possamos percorrer o conteúdo da variável, array, properties. E a cada elemento desse array utilizamos novamente reflexão. Utilizamos primeiro para saber se a propriedade encontrada na posição x possui um Setter, ou seja, se ela tem um Set que permite jogar algum valor para ela. Essa verificação é feita utilizando o método GetSetMethod que retorna um objeto do tipo MethodInfo caso exista um Setter ou NIL caso contrário. Isso significa que não vamos armazenar no cache propriedades somente leitura, pois, se são somente leitura o usuário não pode alterá-las diretamente. Mas se for uma propriedade que permita sua escrita, na sequência armazenamos na variável value o valor da propriedade, recuperada pelo método GetValue e então adicionamos no HashTable um novo item, passando como chave o nome da propriedade, obtido pela propriedade Name e o valor, armazenado em value. Dessa forma percorremos todas as propriedades públicas, de instância e de escrita disponível na instância atual de Self, e armazenamos no HashTable.

O próximo método, CancelEdit vai fazer exatamente o contrário. Irá pegar os valores contidos no HashTable e irá passá-los para as devidas propriedades. Na primeira linha do método, após o Begin, verificamos se nosso cache existe, pois se não existe não há o que recuperar, devolver, para as propriedades e então saímos do método. Na sequência recuperamos novamente a lista de propriedades e armazenamos essa lista em properties. O comando For seguinte é ligeiramente diferente do visto no método BeginEdit. Aqui também verificamos se a propriedade possui um Setter, mas, se possuir recuperamos seu valor do cache passando para ele o nome da propriedade que está na posição x. Esse valor é armazenado em value e então enviado para a propriedade corrente através do método SetValue. Ao terminar de retornar os valores para as propriedades, limpamos o cache, setando para NIL a variável props. Restou-nos apenas o método EndEdit. Este por sua vez é chamado quando finalizamos a edição, e se ela foi finalizada, não precisamos manter um cache com os valores das propriedades, então também setamos para NIL a variável props.

Com tudo isso implementado temos uma classe que serve de base para qualquer entidade.

## Utilizando a classe Entidade e terminando a edição

Para utilizá-la é bem simples, já que todo trabalho foi centralizado nela. Basta alterarmos a declaração das nossas classes de negócio para que fiquem como a seguir, herdando de Entidade:

```
Pessoa = public class(Entidade)
Endereco = public class(Entidade)
Telefone = public class(Entidade)
```

Mas fazer apenas isso não é o suficiente para termos o controle da edição dos objetos. É preciso dizer ao BindingSource corresponde de cada uma que inicie as alterações (antes de alterarmos qualquer dado do objeto), que cancele ou termine essas alterações. O primeiro passo, que seria o de solicitar o início das alterações é feito automaticamente pelo BindingSource. Ao detectar que alguma propriedade de seu Source está prestes a ser alterada, ele dispara o método BeginEdit do Source. Mas para os outros dois métodos nós é que temos a responsabilidade de informar ao BindingSource da sua execução. Então vamos codificar os botões Salvar e Cancelar. O evento Click do botão Salvar fica como mostra a **Listagem 13**.

**Listagem 13**. Confirmando as alterações

```
method EditPessoa.btnSalvar_Click(sender: System.Object; e:
System.EventArgs);
begin
    pessoaBindingSource.EndEdit;
    enderecoBindingSource.EndEdit;
    telefoneBindingSource.EndEdit;
    DialogResult := DialogResult.OK;
end;
```

Nele executamos o método EndEdit de cada BindingSource envolvido e este irá executar o EndEdit de seu Source. Por fim fechamos o formulário alterando sua propriedade DialogResult para OK, indicando que a operação foi encerrada com sucesso. Já na **Listagem 14** podemos ver o evento Click do botão Cancelar.

**Listagem 14.** Cancelando as alterações

```
method EditPessoa.btnCancelar_Click(sender: System.Object; e:
  System.EventArgs);
begin
  pessoaBindingSource.CancelEdit;
  enderecoBindingSource.CancelEdit;
  telefoneBindingSource.CancelEdit;
  DialogResult := DialogResult.Cancel;
end;
```

Este por sua vez chama o método CancelEdit do BindingSource que por sua vez chama o CancelEdit de seu Source. E no final, fechamos o formulário de edição setando sua propriedade DialogResult para Cancel, para indicar que a operação foi cancelada. Ainda não acabou, estamos esquecendo os botões contidos no GroupBox telefones. Eles serão utilizados para adicionar, remover ou editar um dos telefones da pessoa em edição. Vamos codificar primeiramente o botão novo, que pode ser visto na sequência:

```
method EditPessoa.btnNovo_Click(sender: System.Object; e: System.
 EventArgs);
begin
  telefoneBindingSource.AddNew;
  txtNumero.Focus;
end;
```

Nele nós executamos o método AddNew do BindingSource de telefone. Esse método AddNew cria um novo objeto Telefone em seu Source. Então enviamos o foco para o TextBox txt Numero. Já o botão excluir solicita uma confirmação, e sendo confirmado, remove do BindingSource o objeto selecionado:

```
method EditPessoa.btnExcluir_Click(sender: System.Object; e:
   System.EventArgs);
begin
    if MessageBox.Show('Confirma exclusão do telefone selecionado?',
     'Atenção',MessageBoxButtons.YesNo)=DialogResult.Yes then
    begin
        telefoneBindingSource.RemoveCurrent();
    end;
end;
```

O método Cancelar chama o CancelEdit do BindingSource de Telefone, fazendo com que toda alteração seja desfeita:

```
method EditPessoa.btnCancelarTelefone_Click(sender: System.Object;
 e: System.EventArgs);
begin
    telefoneBindingSource.CancelEdit;
end;
```

E por fim o Confirmar confirma a edição do objeto, como a seguir:

```
method EditPessoa.button1_Click(sender: System.Object; e: System.
 EventArgs);
begin
    telefoneBindingSource.EndEdit;
end;
```

A rotina de edição está pronta. Mas, antes de colocar tudo para rodar, falta criarmos a rotina de inclusão de novas Pessoas. Volte ao formulário principal e no botão Nova Pessoa faça como mostra a **Listagem 15**.

**Listagem 15.** Adicionando uma nova pessoa

```
method MainForm.btnNova_Click(sender: System.Object; e: System.
EventArgs);
var
  form: EditPessoa;
begin
  form := new EditPessoa();
  form.PessoaSelecionada := new Pessoa();
  if (form.ShowDialog = DialogResult.OK) then
  begin
      pessoaBindingSource.Add(form.PessoaSelecionada);
  end;
end;
```

Esse código é bem simples de ser entendido. Nele instanciamos o formulário de edição e passamos para sua propriedade PessoaSelecionada uma nova instância de Pessoa. Ou seja, um objeto novo. E chamamos o formulário. Caso o formulário de edição tenha sido encerrado com sucesso (DialogResult.OK), adicionamos ao BindingSource de pessoas o objeto recém criado. Agora sim, nosso protótipo está totalmente funcional. Execute e veja, você poderá consultar objetos, alterá-los ou não, excluir e até mesmo criar novos. Porém, eles não são persistidos. Esse é o nosso próximo passo. Salvar esses objetos em um meio físico.

## Incluindo a persistência

Como já foi mencionado, para salvar nossos objetos em um banco de dados vamos utilizar o Framework NHibernate. A primeira coisa a fazer é baixar o arquivo compactado e descompactá-lo em alguma pasta (veja link para download na seção Links). Antes de começar a aplicar a persistência, vamos entender alguns conceitos. Vejam na **Figura 11** a arquitetura do NHibernate.

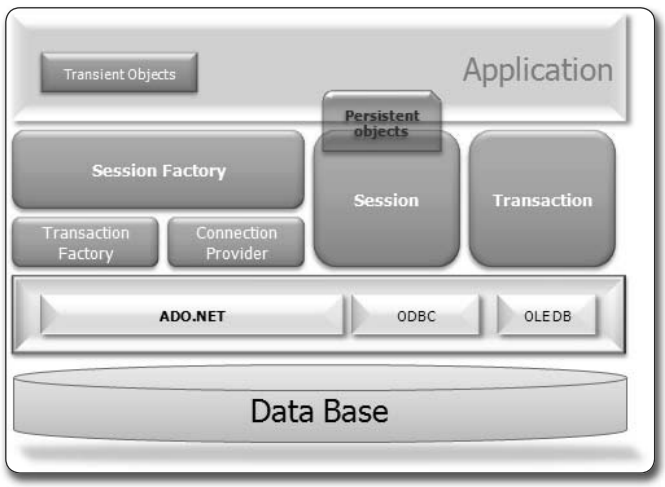



**Figura 11.** Arquitetura do NHibernate

Na camada superior (Application) temos os objetos de negócio definidos por nosso sistema. Esses objetos são enviados para a camada seguinte (NHibernate) que então utiliza os recursos do .NET Framework (ADO.NET, ODBC, OLE DB) para enviar os

dados para um banco de dados. Para fazer isso o NHibernate define algumas estruturas:

**Session Factory**: É o cache com as informações de mapeamento de classes (veja a seguir) e disponibiliza o controle transacional das ações em relação ao banco através da estrutura Transaction Factory. Também disponibiliza através de Connection Provider informações para a realização de uma conexão com banco de dados, que é utilizada pela Session;

**Session**: Representa uma conexão com o banco de dados, que deve ser curta. Nela temos um cache dos objetos que ela contém. Quando um objeto é salvo ou recuperado, suas informações ficam no cache da Session utilizada para essas operações. Oferece meios para se aplicar controle transacional através de Transaction;

**Transaction**: Encapsula uma transação do ADO.NET, permitindo operações atômicas.

De uma forma mais simplista, podemos representar a estrutura de um aplicativo que utiliza NHibernate como mostra a **Figura 12**.

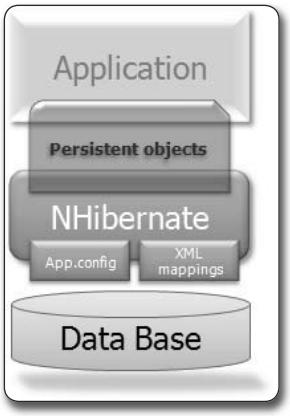



**Figura 12**. Representação básica de um sistema utilizando NHibernate

Sua aplicação possui objetos. Sua aplicação carrega o NHibernate de tal forma que ele lê o mapeamento das classes feito em arquivos XML, lê as configurações do banco de dados definidas também em XML. Então sua aplicação envia para o NHibernate seus objetos e ele os envia para o banco de dados, podendo também fazer a operação inversa.

## Unicidade de um objeto

É sabido que um registro de uma tabela de um banco relacional possui sua unicidade garantida por sua chave primária, seu Id. Chaves fazem parte da estrutura relacional. Porém no mundo orientado a objetos isso não existe. Para intermediar essa diferença o NHibernate oferece mecanismos para gerenciamento de chaves primárias, bastando inicialmente para nós, adicionar uma propriedade em nossas classes que represente esse Id. Então, adicionamos conforme a seguir uma propriedade Id em Entidade:

```
property Id: Int32;
```

E inicializamos no construtor da classe seu valor para zero:

```
constructor Entidade;
begin
  Id := 0;
end;
```

Como todas nossas classes herdam de Entidade, consequentemente todas elas também terão uma propriedade Id do tipo Int32.

## Mapeamento das classes

Tradicionalmente o NHibernate permite que façamos o mapeamento de nossas classes em arquivos XML que devem obedecer a seguinte estrutura em seu nome: <nome_da_classe>.hbm.xml. Vamos começar pelo mapeamento da classe Pessoa. Vale lembrar que é no mapeamento que dizemos como a classe será salva no banco relacional. Para organizar o projeto vamos adicionar uma nova pasta no projeto ModeloAgenda. Para isso clique com o botão direito do mouse sobre o nome do projeto (ModeloAgenda) e selecione Add>New Folder. A pasta será adicionada com um nome padrão, vamos trocar esse nome para Mapeamento. É aí que vamos armazenar os arquivos XML.

**Nota**

O NHibernate disponibiliza outras formas para se realizar esse mapeamento, como custom attributes e através um de um recurso chamado Fluent Mapping.

Para criar um arquivo dentro da pasta, clique com o botão direito sobre ela e escolha Add>New Item. Na janela que surge escolha Data e nela XML File. Corrija o nome do arquivo para Pessoa.hbm.xml. Agora adicione nele o conteúdo da **Listagem 16**.

**Listagem 16**. Mapeamento da classe Pessoa

```
01 <?xml version="1.0" encoding="utf-8" ?>
02 <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
    assembly="ModeloAgenda" namespace="ModeloAgenda" >
03   <class name="Pessoa" table="PESSOA">
04     <id name="Id" column="ID_PESSOA" type="Int32"
         unsaved-value="0">
05       <generator class="hilo"/>
06     </id>
07     <property name="Nome" column="NOME" type="string"
         length="100" not-null="true" />
08     <component name="Endereco" class="Endereco">
09       <property name="Rua" column="RUA" type="string"
          length="100" not-null="false" />
10       <property name="Bairro" column="BAIRRO" type="string"
          length="50" not-null="false" />
11       <property name="Cidade" column="CIDADE" type="string"
          length="80" not-null="false" />
12       <property name="UF" column="UF" type="string"
          length="2" not-null="false" />
13     </component>
14     <bag name="Telefones" table="TELEFONE" generic="true"
        cascade="all-delete-orphan" >
15       <key column="ID_PESSOA" />
16       <one-to-many class="Telefone" />
17     </bag>
18   </class>
19 </hibernate-mapping>
```

Vamos entender o que tudo isso significa. Na Linha 02 estamos especificando que este arquivo é um arquivo de mapeamento e que irá mapear o namespace ModeloAgenda

localizado dentro do assembly ModeloAgenda. Na Linha 03 informamos qual classe é mapeada e para qual tabela do banco de dados ela irá. Nas Linhas 04 à 06 definimos o que é chamado de estratégia de geração de Ids. Nessa parte do arquivo definimos como o NHibernate irá gerar as chaves primárias, que são necessárias para o banco de dados. Existem várias estratégias, explicar cada uma delas foge do objetivo do artigo. Mas adianto que o NHibernate oferece meios para que sejam utilizados recursos específicos de cada banco, como Generators no Firebird e campos Identity no SQL Server, e até mesmo implementação que são independentes de banco de dados. No nosso caso utilizamos uma estratégia chamada HiLo. Nessa estratégia o NHibernate pode criar uma tabela em nosso banco de dados que será utilizada para manter o Id dos objetos de todas as classes ou, manter por classe o Id de cada objeto. Aqui utilizamos da primeira forma.

Quero destacar a Linha 04. Vejam que informamos que temos uma propriedade de nome Id, mas sua coluna no banco de dados será chamada ID_PESSOA. Outra definição importante dessa linha é o valor unsaved-value = 0. Ele serve para indicar ao NHibernate qual objeto ainda não foi salvo no banco de dados. Você lembra que adicionamos um construtor na classe Entidade, e que nele ajustamos a propriedade Id para zero? Muito bem, com isso todo objeto recém criado tem Id = 0 significando que ele ainda não foi salvo no banco dados. Ao especificarmos isso no mapeamento o NHibernate saberá exatamente que é preciso inserir, e não atualizar um registro no banco de dados.

Na Linha 07 vemos o mapeamento de uma propriedade. Informamos o nome da propriedade e seu campo correspondente na tabela. Também foi informado tipo da propriedade segundo os tipos reconhecidos pelo NHibernate (na documentação existente você poderá verificar quais tipos são reconhecidos e como devem ser mapeados). Caso não conste o tipo no arquivo, o NHibernate aplica reflexão na classe que está sendo mapeada para detectar qual o tipo correspondente. Também definimos um tamanho para o campo (Length) e se ele irá rejeitar valores nulos (not-null), essas informações só são utilizadas no momento da criação do banco de dados pelo NHibernate.

Nas Linhas 08 a 13 definimos uma classe componente. Uma classe desse tipo não possui uma tabela própria para armazenar o valor de suas propriedades, mas sim tem suas propriedades mapeadas para campos de outra tabela. Este é mais um caso de impedância. Podemos ter duas classes separadas que são salvas em uma única tabela. É justamente isso que estamos definindo nesse mapeamento. Estamos dizendo que a classe Endereco será salva dentro da classe Pessoa, assim, no banco de dados teremos apenas uma tabela, Pessoa, contendo todos os campos.

Na linha 14 a 17 estamos mapeando o relacionamento de duas classes, Pessoa e Telefone. Vale lembrar que uma pessoa possui N telefones, sendo assim em Pessoa temos uma lista de telefones. Essa lista pode ser mapeada através do tipo bag, como é mostrado. Em name especificamos o nome da propriedade que representa essa lista, no caso Telefones, depois informamos qual tabela possui os dados dessa lista. Indicamos também que essa é mantida utilizando o recurso de Generics e que definimos o comportamento para as situações de cascata. Definimos o valor all-delete-orphan. Esse valor indica ao NHibernate que sempre que adicionarmos um telefone a uma pessoa e salvarmos essa pessoa, esse telefone também deve ser inserido no banco de dados e se excluirmos uma pessoa, seus telefones relacionados também devem ser excluídos. Mas como o NHibernate sabe desse relacionamento? Isso é indicado nas próximas linhas. Na Linha 15 estamos dizendo que a chave de ligação entre as duas tabelas será um campo chamado ID_PESSOA na tabela TELEFONE, e que esse será um relacionamento um para muitos (Linha 16). O NHibernate também oferece suporte para outros relacionamentos existentes no mundo dos bancos relacionais, como M:N, 1:1 e autorrelacionamento.

Agora vamos mapear a classe Telefone. Adicione mais um arquivo XML na pasta Mapeamento, como foi feito anteriormente e acerte seu nome para Telefone.hbm.xml. Insira nele o conteúdo da **Listagem 17**.

**Listagem 17**. Mapeamento da classe Telefone

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
  assembly="ModeloAgenda" namespace="ModeloAgenda" >
  <class name="Telefone" table="TELEFONE">
    <id name="Id" column="ID_TELEFONE" type="Int32"
     unsaved-value="0">
      <generator class="hilo"/>
    </id>
    <property name="Numero" column="NUMERO" type="string"
      length="15" not-null="true" />
    <property name="Ramal" column="RAMAL" type="string"
      length="10" not-null="false" />
  </class>
</hibernate-mapping>
```

De forma similar ao mapeamento da classe Pessoa, também especificamos a estratégia de ID e mapeamos as propriedades. Para finalizar o mapeamento temos que adicionar esses arquivos XML como arquivos de resource (recurso) do assembly ModeloAgenda. Assim não temos que distribuir junto com a aplicação esses XML. Para isso, na Solution Explorer clique com o botão direito no arquivo Pessoa.hbm.xml e escolha Properties. Na janela que abre temos a opção Build Action. Altere essa propriedade para Embedded Resource. Faça o mesmo para o arquivo Telefone.hbm.xml. Assim, quando compilamos o projeto (Ctrl + Shift + B) esses arquivos são adicionados no assembly ModeloPessoa.

## Configurando o acesso ao banco

Mas como o NHibernate sabe qual banco de dados utilizar ou até mesmo onde está o banco de dados. Isso também é definido em um arquivo XML chamado hibernate.cfg.xml. Adicione ao projeto Agenda um novo arquivo XML, defina seu nome para hibernate.cfg.xml e nele insira o conteúdo da **Listagem 18**.

Na Linha 02 indicamos que este é um arquivo de configuração, que será lido juntamente com os arquivos de mapeamento

no momento que carregarmos o NHibernate (veremos isso daqui a pouco). Na Linha 03 damos um nome para esta session factory (configuração de conexão). Da linha 04 a 09 estamos configurando as propriedades básicas de conexão, existem vários parâmetros mas estamos utilizando apenas o básico:

Linha 04 - dialect: Indica que deve ser utilizado SQL específico para o banco de dados Firebird. Ou seja, o NHibernate poderá gerar SQL específico para os recursos do Firebird, obedecendo sua sintaxe;

Linha 05 – connection.driver_class: Informa qual provider utilizar para que se realize a conexão entre o ADO.NET e o banco de dados Firebird. No caso especificamos que seja utilizado o ADO.NET Provider disponibilizado no site do Firebird;

Linha 06 – connection.connection_string: É a string de conexão com o banco de dados. Definimos a senha, o caminho do banco etc.

Linha 07 – connection.isolation: Indica qual o nível transacional será empregado pelo ADO.NET. Neste caso utilizamos o ReadCommited, que é o mais indicado na maioria dos casos;

Linha 08 – proxyfactory.factory_class: Informa qual Framework será utilizado para geração de objetos Proxy. Esses objetos permitem ao NHibernate realizar operações de otimização. Uma delas é chamada de Lazy Loading, ou carregamento tardio. Para exemplificar sua importância vamos analisar nossa regra de negócio. Temos uma classe pessoa que possui um relacionamento com a classe Telefone. Uma pessoa pode ter N telefones. Imagine que ao fazermos uma pesquisa de pessoa, 100.000 objetos pessoa sejam recuperados. Será que seus telefones não serão carregados juntos também? Imagine que cada pessoa tenha 3 telefones, serão 300.000 objetos a mais na memória. O Lazy Loading faz com que isso não aconteça. Ele permite que a lista de telefones de uma pessoa seja carregada apenas quando for acessada. Isso possui seus prós e contras, mas é uma discussão para um próximo artigo;

Linha 09 – use_proxy_validator: Uma das implicações em se utilizar o Lazy Loading é ter que definir todas as propriedades das classes como Virtual. O NHibernate possui um método que verifica isso no momento em que é inicializado. Essa verificação pode consumir um tempo caso haja muitos arquivos de mapeamento, então podemos habilitar ou não essa verificação. Já na Linha 10 estamos informando ao NHibernate em que assembly estão os arquivos de mapeamento para que possam ser extraídos e lidos. Agora, clique com o botão direito sobre o arquivo, na Solution Explorer, e escolha Properties. Ajuste sua propriedade Copy to Output Directory para Always, assim, sempre que compilarmos o projeto o arquivo será copiado para a pasta do programa.

## Iniciando o NHibernate

Anteriormente citei algumas estruturas que compõem o núcleo do NHibernate. Dentre elas, a que consome mais recursos é a SessionFactory. Instanciá-la é custoso porque é nela que o NHibernate realiza suas tarefas mais básicas, porém de grande importância. Ao criar uma SessionFactory o NHibernate procura na pasta do executável o arquivo de configuração (hibernate.cfg.xml) para que saiba onde está o banco de dados, qual banco de dados será utilizado e qual assembly contém o mapeamento. De posse dessas informações, através de reflexão, os arquivos XML armazenados como embedded resources são extraídos para memória e então passam por uma validação de sintaxe. Tudo isso custa tempo de processamento e memória. Então a principal recomendação do time do NHibernate é: Se seu sistema utiliza apenas um banco de dados, você deve ter apenas uma SessionFactory instanciada durante o uso do sistema. É nossa obrigação criar um mecanismo que gerencie isso para nós. Para garantirmos que haja apenas uma instância de um objeto, vamos aplicar um padrão de projeto chamado Singleton.

**Listagem 18**. Configurando o acesso ao banco de dados

```
01 <?xml version="1.0" encoding="utf-8" ?>
02 <hibernate-configuration  xmlns="urn:nhibernate-configuration-2.2" >
03   <session-factory name="Agenda">
04     <property name="dialect">NHibernate.Dialect.FirebirdDialect
        </property>
05     <property name="connection.driver_class">NHibernate.
        Driver.FirebirdClientDriver</property>
06     <property name="connection.connection_string">
         User=SYSDBA;Password=masterkey;Database=C:\Paulo\
          ClubeDelphi\Artigos\DelphiPrism - Final\Banco\
           PESSOAS.FDB;DataSource=localhost;Port=3060;
           Dialect=3;Charset=NONE;Role=;Connection lifetime=15;
            Pooling=true;MinPoolSize=0;MaxPoolSize=50;Packet
             Size=8192;ServerType=0;</property>
07     <property name="connection.isolation">ReadCommitted
        </property>
08     <property name="proxyfactory.factory_class">NHibernate.
         ByteCode.Castle.ProxyFactoryFactory, NHibernate.
          ByteCode.Castle</property>
09     <property name="use_proxy_validator">false</property>
10     <mapping assembly="ModeloAgenda"/>
11   </session-factory>
12 </hibernate-configuration>
```

Selecione o projeto Agenda e nele crie uma nova pasta chamada Util, da mesma forma como fizemos para criar a pasta Mapeamento. Agora adicione nessa pasta uma nova unit (classe) chamada NHibernateHelper.pas. Nesta classe vamos criar a rotina que irá garantir que apenas uma SessionFactory seja criada. Agora, nessa nova classe adicione o código da **Listagem 19**.

Na seção uses temos as referências que precisam ser adicionadas. Na sequência (Linha 14) definimos uma classe chamada NHibernateHelper. Nela definimos na seção privada um campo estático do tipo ISessionFactory, um método estático que retorna um ISessionFactory e uma propriedade estática que utiliza o método GetSessionFactory para retornar um ISessionFactory. Depois na seção public temos um método estático que retorna um ISession. Veja a Linha 40, ela é a implementação do método Open Session. Nele estamos acessando a propriedade SessionFactory e solicitamos a ela que abra uma sessão para nós. Por sua vez a propriedade SessionFactory utiliza o método GetSessionFactory. Vamos entendê-lo.

Na Linha 28 verificamos se o campo estático FSessionFactory é nil, ou seja, se ainda não foi instanciado. Se ainda não foi, nós então carregamos o NHibernate. Instanciamos um objeto do tipo Configuration e na sequência executamos seu método Configure. É aqui que os arquivos XML são localizados e validados. Depois instanciamos outro objeto do tipo SchemaExport e executamos seu método Execute. Esse método permite que o

esquema do banco de dados seja criado a partir dos arquivos de mapeamento. É possível também apenas exportar esse esquema gerado, aqui o configuramos para criar a estrutura em um banco previamente criado. Então por fim retornamos esse campo estático FSessionFactory. Caso esse campo já tenha sido criado, ele não é instanciado novamente, garantindo assim que haja apenas uma ISessionFactory durante a execução do sistema.

Para realizar qualquer operação de acesso a banco de dados é necessário termos uma sessão disponível (ISession). Essa sessão representa seu acesso ao banco de dados. Sendo assim, o fluxo padrão para salvar alguma informação (objeto) em um banco de dados utilizando NHibernate pode ser representado como mostra a **Figura 13**.

**Listagem 19**. NHibernateHelper

```
01 namespace Agenda.Util;
02
03 interface
04
05 uses
06   System.Collections.Generic,
07   System.Linq,
08   NHibernate,
09   NHibernate.Cfg,
10   NHibernate.Tool.hbm2ddl,
11   System.Text;
12
13 type
14   NHibernateHelper = public class
15   private
16       class FSessionFactory : ISessionFactory;
17       class method GetSessionFactory: ISessionFactory;
18       class property SessionFactory: ISessionFactory read
          GetSessionFactory;
19   protected
20   public
21       class method OpenSession:ISession;
22   end;
23
24 implementation
25
26 class method NHibernateHelper.GetSessionFactory: ISessionFactory;
27 begin
28     if (FSessionFactory = nil) then
29     begin
30      var configuration := new Configuration();
31      configuration.Configure();
32      new SchemaExport(configuration).Execute(false, true, false);
33      FSessionFactory := configuration.BuildSessionFactory();
34     end;
35     result := FSessionFactory;
36 end;
37
38 class method NHibernateHelper.OpenSession: ISession;
39 begin
40     result := SessionFactory.OpenSession;
41 end;
42
43 end.
```

**Figura 13**. Fluxo de operação

No primeiro passo (Abrir sessão) é que vamos fazer uso de nosso NHibernateHelper, para que ele devolva para nós uma sessão válida. Então, a partir dessa sessão obtida podemos iniciar uma transação e enviar através da sessão uma alteração ao banco de dados onde, ocorrendo tudo bem, comitamos a transação e fechamos a sessão. Vamos ver isso na prática agora.

## Salvando um objeto

A primeira coisa a se fazer é adicionar algumas referências ao projeto Agenda, como mostra a **Figura 14**.

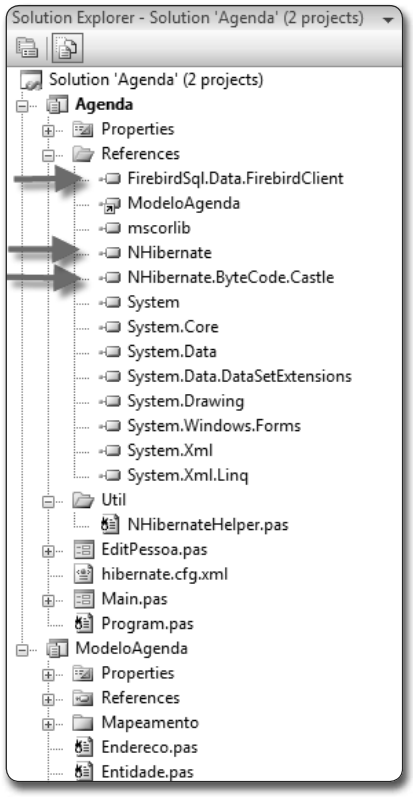



**Figura 14**. Referências necessárias

Posteriormente crie um banco de dados Firebird, vazio, sem nada, em alguma pasta e acerte no arquivo hibernate.cfg.xml o caminho do banco de dados, veja **Listagem 18**. Agora vamos abrir o formulário Main.pas. Remova de seu construtor as rotinas que criavam objetos para testes. Feito isso, dê um clique duplo do botão Nova Pessoa e ajuste seu código para que fique como a **Listagem 20**.

Da Linha0 1 à Linha 07 temos o código que tínhamos antes. Mas vejam a Linha 09. Nela usamos o comando using em conjunto ao nosso NHibernateHelper para abrir uma sessão.

Na sequência já temos uma sessão disponível armazenada na variável "s" e solicitamos a ela que inicie uma transação, que ficará armazenada na variável tx. Depois iniciamos um bloco try/except e na Linha 14 pedimos a nossa sessão, representada pela variável s, que salve o objeto PessoaSelecionada. Esse salvar é realizado pelo método SaveOrUpdate. Esse método identifica a situação do objeto, inicialmente verificando se seu Id é zero, para então inserir um registro

no banco de dados. Caso o Id do objeto não seja zero, será executada uma sentença de atualização (update). Estando tudo certo, comitamos a transação e com isso o objeto salvo vai para o cache da sessão em uso. Mas, caso algo dê errado nós efetuamos um Rollback na transação e limpamos o cache da sessão utilizando o método Clear. Perceberam como foi simples salvar um objeto? Em nosso código não fazemos menção alguma a banco de dados ou SQL. A edição de um objeto é também praticamente igual, como mostra a **Listagem 21**.

**Listagem 20**. Salvando um objeto

```
01 method MainForm.btnNova_Click(sender: System.Object; e:
    System.EventArgs);
02 var
03     form: EditPessoa;
04 begin
05   form := new EditPessoa();
06   form.PessoaSelecionada := new Pessoa();
07   if (form.ShowDialog = DialogResult.OK) then
08   begin
09       using s:= NHibernateHelper.OpenSession do
10       begin
11         using tx := s.BeginTransaction do
12         begin
13           try
14             s.SaveOrUpdate(form.PessoaSelecionada);
15             tx.Commit;
16             pessoaBindingSource.Add(form.PessoaSelecionada);
17           except
18               on ex:Exception do
19               begin
20                 tx.Rollback;
21                 s.Clear;
22                 MessageBox.Show('Falha ao editar pessoa.
                    Mensagem original: '+ ex.Message);
23               end;
24           end;
25         end;
26       end;
27   end;
28 end;
```

**Listagem 21**. Salvando a edição realizada em um objeto

```
method MainForm.btnEditar_Click(sender: System.Object; e:
 System.EventArgs);
var
    form: EditPessoa;
begin
    form := new EditPessoa();
    form.PessoaSelecionada := (pessoaBindingSource.Current as
    Pessoa);
    if (form.ShowDialog = DialogResult.OK) then
    begin
      using s:= NHibernateHelper.OpenSession do
      begin
        using tx := s.BeginTransaction do
        begin
          try
            s.SaveOrUpdate(form.PessoaSelecionada);
            tx.Commit;
          except
              on ex:Exception do
              begin
                tx.Rollback;
                s.Clear;
                MessageBox.Show('Falha ao editar pessoa.
                 Mensagem original: '+ ex.Message);
              end;
          end;
        end;
      end;
    end;
end;
```

## Excluindo um objeto

A exclusão também é prática como o salvar. Dê um clique duplo sobre o botão Excluir e altere sua implementação para que fique como mostra a **Listagem 22**.

### Nota do DevMan

O comando using tem a funcionalidade combinada do comando with e do bloco try/finally para liberar da memória o objeto em uso. Sendo assim, utilizar:

```
using f := new FileStream(...) do begin
  ...
end;
```

É o mesmo que:

```
with f := new FileStream(...) do try
  ...
finally

  if (f is Idisposable) then (f as IDisposable).Dispose();
end;
```

**Listagem 22**. Excluindo um objeto

```
01 method MainForm.btnExcluir_Click(sender: System.Object; e:
    System.EventArgs);
02 begin
03     if MessageBox.Show('Confirma exclusão da Pessoa
        selecionada?','Atenção',MessageBoxButtons.YesNo)=
        DialogResult.Yes then
04     begin
05       using s:= NHibernateHelper.OpenSession do
06       begin
07         using tx := s.BeginTransaction do
08         begin
09           try
10             s.Delete(pessoaBindingSource.Current as Pessoa);
11             tx.Commit;
12             pessoaBindingSource.RemoveCurrent();
13          except
14               tx.Rollback;
15               s.Clear;
16           end;
17         end;
18       end;
19     end;
20 end;
```

Após solicitar uma confirmação ao usuário, de forma semelhante nós abrimos uma sessão (Linha 05), iniciamos uma transação (Linha 07) e adicionamos um bloco try/except (Linha 09). Dentro desse bloco solicitamos à sessão que exclua o objeto que está selecionado no DataGrid, representado pela propriedade Current de pessoaBindingSource. Essa exclusão é feita chamando o método Delete de ISession. Caso dê certo, comitamos a transação e removemos o objeto da lista, mas se algo dá errado, voltamos à transação (rollback) e limpamos o cache da sessão. Agora só falta implementar a rotina de busca de objetos.

## Localizando um objeto

Crie um novo método privado, como mostra a **Listagem 23**.

**Nota**

Adicione à seção uses uma referência ao namespace NHibernate.Criterion.

Na Linha 05 e 07 abrimos uma sessão e iniciamos uma transação, como feito anteriormente. Mas veja a Linha 09. Estamos utilizando um método de busca oferecido pela sessão do NHibernate, o Criteria. Esse método de busca

permite que criemos uma consulta de forma orientada a objetos. Ela oferece métodos que nos permitem criar filtros para localização. Veja na Linha 10. Estamos utilizando um Like para realizar uma busca parcial pelos nomes. Ao executar o método List<T> do Criteria, esses filtros são transformados em SQL e são enviados para o banco de dados. O retorno que o banco gerar, será transformado de volta em objetos do tipo Pessoa e serão armazenados em forma de lista na variável resultados. Então na sequência (Linha 13) instanciamos nossa propriedade ListaPessoas, passando para ela a lista de resultados e definimos que ela é o DataSource de pessoaBindingSource. Assim, caso exista algum retorno, ele será exibido no DataGrid. O NHibernate oferece outros mecanismos de buscas, como o HQL que é uma linguagem de consulta, muito similar ao SQL mas que não utiliza campos e sim objetos e suas propriedades para consulta.

**Listagem 23**. Método de busca de objetos

```
01 method MainForm.Localizar(Nome: String);
02 var
03   resultados: IList<Pessoa>;
04 begin
05     using s:= NHibernateHelper.OpenSession do
06     begin
07         using tx := s.BeginTransaction do
08         begin
09           resultados := s.CreateCriteria<Pessoa>
10            .Add(Restrictions.Like('Nome',Nome,MatchMode.Start))
11            .List<Pessoa>;
12
13           ListaPessoas := new BindingList<Pessoa>(resultados);
14           pessoaBindingSource.DataSource := ListaPessoas;
15           tx.Commit;
16         end;
17     end;
18 end;
```

## Executando o programa

Ao executar o programa o NHibernate ainda não é carregado. Isso vai acontecer quando tentarmos abrir nossa primeira sessão, lembre-se do funcionamento de nosso NHibernateHelper. Em consequência disso, o primeiro acesso ao banco de dados será mais lento que os acessos subsequentes. Ao executar o programa, clique inicialmente no botão Localizar. Esse por sua vez vai carregar o NHibernate e esse irá criar o banco de dados para nós por causa da chamada ao método Execute da classe SchemaExport (Linha 32 da **Listagem 19**). Veja nas **Figuras 15, 16** e **17** as tabelas criadas.

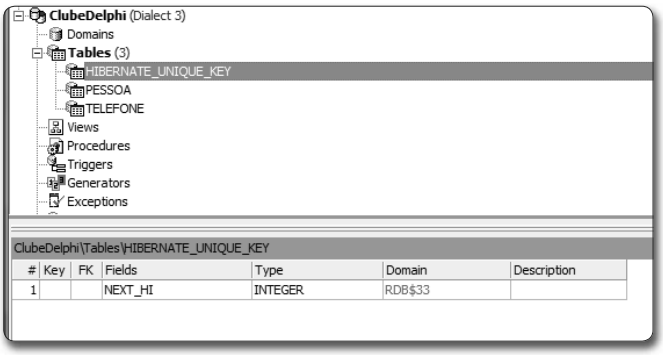



**Figura 15**. Tabela para controle de geração de Id

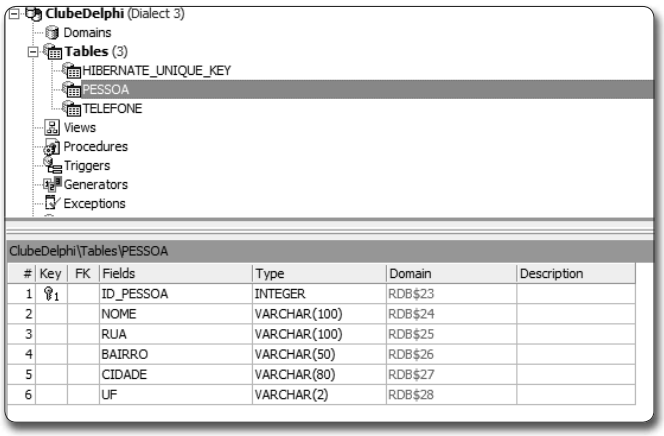



**Figura 16**. Tabela PESSOA

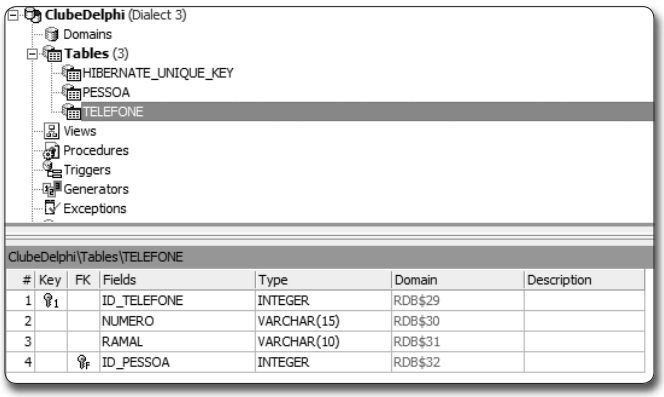



**Figura 17**. Tabela TELEFONE

Observe que as tabelas PESSOA e TELEFONE foram criadas obedecendo as informações contidas nos arquivos de mapeamento e inclusive na tabela TELEFONE foi criada uma chave estrangeira apontando para PESSOA. Cadastre algumas pessoas, veja que os telefones são salvos de forma automática. Localize também algumas pessoas e veja que os telefones também são trazidos automaticamente, não é mais necessário se preocupar com relacionamentos mestre-detalhe.

## Conclusão

Se vocês perceberam bem, aplicar a persistência de objetos usando o NHibernate foi bastante simples. A parte mais complicada, e que exige um certo estudo, é a tarefa de realizar o mapeamento. Porém este sendo feito, ganhamos uma produtividade enorme, já que aquelas tarefas repetitivas de gerar SQL não estarão mais sobre nossas costas, isso será obrigação do NHibernate. Além disso, vocês notaram que nós alteramos apenas o formulário principal? Não foi preciso alterar nada no formulário de edição. Neste artigo contemplamos apenas uma casca dos recursos oferecidos pelo NHibernate, como por exemplo a possibilidade de trocar de banco de dados sem alterar nenhum código do seu sistema, apenas especificando no arquivo nhibernate.cfg.xml o novo banco de dados, o uso de Projections, Joins entre classes, o próprio uso da linguagem HQL e muito mais.

Espero que este artigo tenha servido para abrir a mente de vocês para outras possibilidades além da velha e querida dupla dbExpress e ClientDataSet e que ao menos desperte a curiosidade sobre o assunto prático e real da orientação a objetos. ●

**Links**

**Site do NHibernate, download, fórum e documentação**
*www.nhforge.org*

**Download Delphi Prism**
*https://downloads.embarcadero.com/free/delphi_prism*

**Blog de um dos desenvolvedores chefe do NHibernate**
*http://www.fabiomaulo.blogspot.com/*

**Blogs do autor**
*http://pauloquicoli.spaces.live.com*
*http://twitter.com/pauloquicoli*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

**Para Delphi 2009/2010**

Do premiado autor da série
**Dominando o Delphi: a Bíblia**

Domine todos os novos recursos do Delphi, como o suporte a Unicode, os tipos genéricos, os métodos anônimos, os controles Ribbon (e outros novos controles), o DBExpress 4, o novo DataSnap e as novas funcionalidades do IDE.

Os novos recursos do Delphi são abordados em profundidade, com inúmeros exemplos de como são usados na prática. Os problemas que podem ser encontrados ao atualizar programas escritos em versões anteriores do Delphi (no Delphi 7 ou Delphi 2007, por exemplo), em especial devido à introdução do suporte a Unicode, são discutidos em detalhes.

*Recursos Avançados do Delphi* é indispensável para todos os desenvolvedores, mesmo iniciantes, já que apresenta informações que não são encontradas em outros livros, e que são necessárias para escrever programas que façam proveito pleno dos novos recursos do Windows XP, Vista e 7.

**416 páginas – lançamento em 09/2009 – R$ 79,00**

Nas melhores livrarias ou em
**www.infopress.com.br**

**Nesta seção você encontra artigos sobre técnicas que poderão aumentar a qualidade do desenvolvimento de software**

# Desenvolvendo aplicativos com suporte a Plugins

## Aprenda como modificar sistemas sem alterar uma única linha de código

**Do que trata o artigo**
Aprenda a criar sistemas com uma arquitetura orientada a Plugins, permitindo a expansão das funções originais do seu sistema sem que seja necessário alterar uma única linha do programa. Além disto, você também aprenderá a criar DLL's com o Delphi 2010.

**Para que serve**
Sistemas com arquitetura orientada a Plugins conseguem diminuir a dificuldade de atualização do software, pois se torna necessário trocar apenas a DLL do Plugin que foi alterado. Surge assim uma aplicação mais flexível e modular o que consequentemente facilita a manutenção do sistema.

**Em que situação o tema é útil**
Você é quem decide onde e como os Plugins podem ser encaixados (integrados), em sua aplicação. O limite é sua própria imaginação.

### Resumo do DevMan

De um modo geral, um Plugin é algo que pode ser integrado em uma aplicação para expandir as funções originais da mesma. A princípio temos a impressão de criar um programa que permita a expansão das funções através de Plugins seja uma tarefa complicada, porém, este artigo mostra como pode ser simples criar este tipo de arquitetura, mesmo para aqueles que não tenham grande experiência com a linguagem.

**Rafael Stavarengo**
*faelsta@gmail.com*
*http://www.devmedia.com.br/stavarengo*
*É programador-analista da Cheina Informática. Trabalha com Delphi, JME e PHP + ZendFramework. Formando em Tecnologia de Análise e Desenvolvimento de Sistemas pela UNIPAR.*

Sistemas orientados a Plugins são comuns. Não é necessário pensar muito para lembrar de programas que permitem a expansão de suas funções através dos mesmos. Até mesmo a própria IDE do Delphi permite a instalação de Plugins. Para entender melhor pense no Firebird. Através de uma arquitetura de Plugins, você consegue fazer com que ele entenda novas funções: para isto você precisar criar uma DLL, que será registrada (plugada)

ao SGBD, expandindo as funções originais do Firebird. Outro exemplo de uso muito simples de entender é o Framework DBExpress. Com o DBExpress você pode adicionar suporte a novos bancos de dados, apenas registrando uma nova DLL referente ao banco desejado. Em outras palavras, o DBExpress pode trabalhar com diversos bancos de dados sem a necessidade de atualizar o componente, ou o Delphi, graças ao uso de Plugins.

Para criar uma arquitetura de Plugins, um sistema normalmente utiliza DLL's (Dynamic Link Library). As DLL's nada mais são do que executáveis cujos métodos podem ser executados por outro executável. Então para criar um sistema expansível através de Plugins você precisa apenas de um arquivo DLL que tenha os novos métodos que sua aplicação irá executar. É claro que para isto é necessário que exista um padrão para os nomes dos métodos que existirem dentro da DLL. Falaremos disto em outro momento.

Criar uma arquitetura de Plugins pode tornar seu sistema mais fácil de atualizar, pois o usuário precisa atualizar apenas a DLL que sofreu a atualização. Além disto, ao criar uma aplicação que utiliza Plugins você também está criando uma aplicação modular e flexível, pois muitas partes do código que podem variar, serão encapsuladas dentro de DLL's. Outra vantagem interessante da utilização de Plugins é o fato de que outros programadores, sejam eles da sua equipe de desenvolvimento ou não, poderão criar DLL's para sua aplicação, independente da linguagem em que programam. Tudo que é necessário saber é o nome dos métodos que devem existir dentro da DLL para que sua aplicação faça a chamada corretamente.

## Entendendo a arquitetura

Para explicar como pode ser criada uma arquitetura de Plugins vou apresentar uma aplicação de exemplo que simula um cadastro. A aplicação vai permitir o cadastro de informações sobre contatos como nome, endereço e CEP. Também será possível excluir um contato já cadastrado.

Quando você estiver criando uma arquitetura de Plugins para sua aplicação você precisa deixar bem claro qual o objetivo do mesmo, para que o desenvolvedor que irá criar os Plugins (as DLL's), não fique em dúvida quanto ao que ele está criando.

Após concluir a parte de salvar e excluir um contato, o próximo passo será criar os Plugins que poderão interagir com estas ações. A ideia é que a aplicação de exemplo possa interagir com as DLL's através de dois métodos que serão chamados pela aplicação depois de salvar ou excluir um registro. Este será o objetivo do nosso Plugin, interagir com o cadastro da aplicação. Então Além da aplicação propriamente dita, deverá existir ao menos um arquivo DLL que será localizado e carregado pela aplicação para que os métodos possam ser invocados quando for o momento apropriado. É importante destacar também que o aplicativo não deve gerar exceções se nenhuma DLL (Plugin) existir.

Você pode definir um diretório padrão onde sua aplicação irá procurar as DLL's, como faz o Firebird por exemplo. No Firebird é recomendado que as DLL's (UDF's) sejam armazenadas dentro da pasta UDF do servidor. Neste exemplo as DLL's devem ser colocadas na mesma pasta onde o EXE do programa será instalado.

Para que os Plugins tenham motivo para existir é necessário que exista uma aplicação para usá-los, por isto, vou começar por ela. No Delphi 2010 crie uma nova aplicação através do menu File>New>VCL Forms Applications. Em seguida adicione e configure os componentes conforme a **Tabela 1**.

| Componente | Propriedade | Valor |
|---|---|---|
| TButton | Name | BotaoAdicionar |
| | Caption | Adicionar contato |
| TButton | Name | BotaoExcluirContato |
| | Caption | Excluir contato |
| TButton | Name | BotaoProcurarDll |
| | Caption | Procurar e registrar os plugins encontrados |
| TEdit | Name | Nome |
| TEdit | Name | Endereço |
| TEdit | Name | CEP |
| TListBox | Name | ListaDeContatos |
| TLabel | Caption | Nome |
| TLabel | Caption | Endereço |
| TLabel | Caption | CEP |
| TLabel | Caption | Contatos existentes |

**Tabela 1**. Propriedades dos componentes

Após criar e configurar as propriedades dos componentes, tente deixá-los como estão apresentados na **Figura 1**.

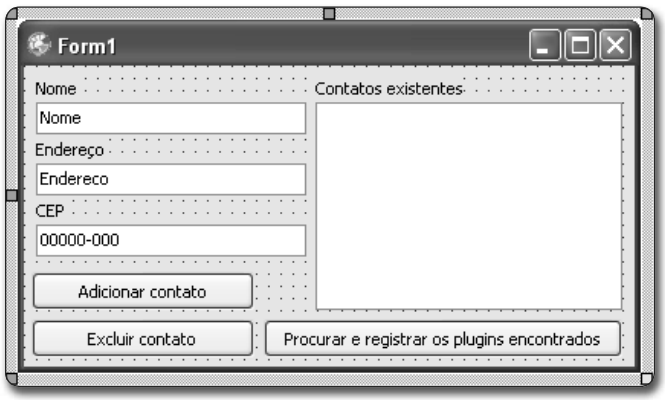



**Figura 1.** Layout do formulário principal

A **Figura 1** mostra a janela de cadastro de contatos da aplicação. A intenção é bem simples. A aplicação deve ser capaz de salvar e excluir contatos, além de ser capaz de expandir estas funções através de Plugins. Ao invés de salvar os contatos em um banco de dados, os contatos serão adicionados na lista virtual do ListBox apresentado na **Figura 1**. Eu disse lista virtual porque quando a aplicação for fechada a lista será perdida. O objetivo aqui não é focar no tema BD. Antes de ver como os Plugins serão registrados, implemente os eventos OnClick dos botões de salvar e excluir contatos. Os códigos dos botões podem ser vistos na **Listagem 1**.

**Listagem 1**. Os métodos de salvar e excluir contatos

```
01 var
02  Form1: TForm1;
03 implementation
//Adicione ao uses
04 uses Types, IOUtils;
05{$R *.dfm}
06 procedure TForm1.BotaoAdicionarClick(Sender: TObject);
07 var
08  Aux: string;
09  I: Integer;
10 begin
11  Aux := Nome.Text + ' - ' + Endereco.Text + ' - ' + Cep.Text;
12  ListaDeContatos.Items.Add(Aux);
13  for I := 0 to GetPlugins.Count - 1 do
14  begin
15    GetPlugins[i].AfterSave;
16  end;
17 end;
18 procedure TForm1.BotaoExcluirContatoClick(Sender: TObject);
19 var
20  I: Integer;
21 begin
22  if ListaDeContatos.Count > 0 then
23    ListaDeContatos.DeleteSelected;
24  for I := 0 to GetPlugins.Count - 1 do
25  begin
26    GetPlugins[I].AfterRemove;
27  end;
28 end;
```

**Nota**

Além das units declaradas na sessão uses da linha 4, você também deverá adicionar as units PluginAdapter e Generics.Collections na sessão uses da Unit1. Generics.Collections faz parte do Delphi, porém, a unit PluginAdapter não. Em três parágrafos você verá do que se trata esta unit.

Como já era esperado, existem poucas novidades no código da **Listagem 1**. Entre as Linhas 06 e 17, você pode ver como os contatos são adicionados à lista e entre as Linhas 18 e 28 está a declaração do método para excluir os contatos da lista. Os códigos destes métodos são simples e dispensam explicações, exceto pelo código que pode ser visto entre as Linhas 13 e 16 e entre as Linhas 24 e 27. O código existente entre estas Linhas tem o mesmo objetivo. Eles interagem com a lista de Plugins encontradas no diretório onde o EXE está instalado e para cada Plugin encontrado é chamado método da DLL correspondente à ação executada.

Como havia dito, a aplicação deve ser capaz de expandir os métodos de adicionar e excluir contatos através de Plugins. Para conseguir isto eu defini dois métodos que as DLL's de Plugins devem implementar, são eles: AfterSave e AfterRemove. Em outras palavras, toda DLL de Plugin para esta aplicação deve possuir estes dois métodos. É você quem escolhe os nomes dos métodos que as DLL's devem possuir. Eles serão chamados pela aplicação respectivamente após salvar e excluir um contato. O código GetPlugins[I] que pode ser visto nas Linhas 15 e 26 da **Listagem 1** retorna um objeto da classe TPluginAdapter. Esta classe representa um Plugin (uma DLL), do sistema. A classe TPluginAdapter faz o que pode ser chamado de 'trabalho sujo'. Ela é classe que sabe como fazer chamadas aos métodos AfterSave e AfterRemove que estão na DLL.

Como você deve imaginar, devem ser feitas algumas considerações antes de fazer chamadas a métodos de uma DLL. Primeiro é necessário carregar a DLL para memória e depois de usá-la é necessário descarregá-la da memória para poupar os recursos de hardware. Além disto, pode não haver DLL para se fazer chamadas e esta situação deve ser considerada. Existe ainda a possibilidade de que a DLL encontrada não possua os métodos que a aplicação vai chamar. Você não vai querer que todos estes testes e considerações estejam espalhadas pela aplicação inteira, por isto existe a classe TPluginAdapter. Esta classe é responsável por todo o tratamento que deve ser feito antes de chamar um método da DLL.

## Adaptando uma DLL

Na **Listagem 3** está o 'cabeçalho' do adaptador, a classe TPluginAdapter. Tenha em mente que esta classe existe apenas para facilitar o trabalho de se fazer chamadas aos métodos de uma DLL. Ela realmente funciona como um adaptador, pois faz toda conversão ou adaptação necessária pra que a aplicação trabalhe com os Plugins como se estivesse fazendo chamadas a métodos de um objeto, quando na verdade, internamente a classe TPluginAdapter está executando os métodos de um arquivo DLL. Veja nas Linhas 11 e 12 da **Listagem 2** a declaração dos métodos que serão usados para carregar e descarregar um arquivo DLL da memória, fazendo com que seja possível usar a DLL. Estes métodos (Linhas 11 e 12) estão declarados na sessão private, pois serão chamados automaticamente pelo adaptador quando for necessário. O arquivo DLL que será adaptado para se parecer com um objeto, é definido durante a criação do objeto de adaptação, veja na Linha 14 que o método construtor recebe o caminho de um arquivo DLL que será adaptado. O caminho desta DLL pode ser recuperado através do método GetDllName declarado na Linha 18. Além destes métodos, como já era de se esperar o adaptador possui a declaração dos métodos AfterSave e AfterRemove que todas as DLL's de Plugin devem ter. Veja na **Listagem 3** como o adaptador faz a chamada aos respectivos métodos das DLL's.

**Nota**

Os métodos LoadLibrary e GetProcAddress que podem ser vistos respectivamente nas Linhas 19 e 30 da **Listagem 3** e o método FreeLibrary (que será mostrado mais adiante) são métodos que estão na unit Windows, por isto não se esqueça de fazer a declaração desta unit na sessão uses conforme aparece na Linha 3 da **Listagem 2**.

O primeiro método apresentado na **Listagem 3** é o método LoadPlugin. Este será responsável por carregar a DLL definida no construtor da classe para a memória. Somente após seu carregamento será permitida a realização da chamada aos métodos que existem dentro dela. Em seguida nas Linhas 13 e 24 podemos observar a implementação dos métodos AfterRemove e AfterSave respectivamente. Lembre-se de que estes métodos são a implementação do adaptador, ou seja, a **Listagem 3** não apresenta o código do Plugin propriamente dito, ao invés disto está apresentando o código de um adaptador que fará a chamada dos métodos existentes dentro do Plugins. Como você pode ver nas Linhas 17 e 28, primeiro é verificado se a DLL já foi carregada para a memória. Se

LoadPlugin retornar verdadeiro é usada a função GetProcAddress para pegar o ponteiro do método desejado dentro da DLL. Você pode ver como o método GetProcAddress é usado nas Linhas 19 e 30. O primeiro parâmetro passado no método é uma variável do tipo THandle. Está variável está declarada na Linha 9 da **Listagem 2** e um valor é atribuído a ela no método LoadPlugin com pode ser visto na Linha 6 da **Listagem 3**. O segundo parâmetro deve ser o nome do método cujo ponteiro será retornado. Se o método procurado não existir dentro da DLL, GetProcAddress irá retornar nil e o teste feito nas Linhas 20 e 31 retornará false fazendo com que nada aconteça. Se a DLL possuir o método procurado então os testes das Linhas 20 e 31 irão retornar true e os respectivos métodos serão executados, como pode ser visto nas Linhas 21 e 32.

**Listagem 2**. A interface do adaptador

```
01 unit PluginAdapter;
02 interface
03 uses Windows;
04 type
05  TAfterSave = procedure; stdcall;
06  TAfterRemove = procedure; stdcall;
07  TPluginAdapter = class
08  private
09    DllHandle: THandle;
10    DllFileName: string;
11    function LoadPlugin: Boolean;
12    procedure FreePlugin;
13  public
14    constructor Create(DllFileName: string);
15    destructor Destroy; override;
16    procedure AfterSave;
17    procedure AfterRemove;
18    function GetDllName: string;
19  end;
20 implementation
```

**Listagem 3**. A implementação do adaptador

```
01 function TPluginAdapter.LoadPlugin: Boolean;
02 begin
03  if DllHandle = 0 then
04  begin
05    try
06      DllHandle := LoadLibrary(PChar(GetDllName));
07    except
08      DllHandle := 0;
09    end;
10  end;
11  Result := DllHandle <> 0;
12 end;
13 procedure TPluginAdapter.AfterRemove;
14 var
15  MetodoAfterRemove: TAfterRemove;
16 begin
17  if LoadPlugin then
18  begin
19    MetodoAfterRemove := GetProcAddress(DllHandle, 'AfterRemove');
20    if Assigned(MetodoAfterRemove) then
21      MetodoAfterRemove;
22  end;
23 end;
24 procedure TPluginAdapter.AfterSave;
25 var
26  MetodoAfterSave: TAfterSave;
27 begin
28  if LoadPlugin then
29  begin
30    MetodoAfterSave := GetProcAddress(DllHandle, 'AfterSave');
31    if Assigned(MetodoAfterSave) then
32      MetodoAfterSave;
33  end;
34 end;
```

Repare nas Linhas 15 e 26 a existência de duas variáveis com os tipos TAfterRemove e TAfterSave onde os ponteiros para os métodos são armazenados (Linhas 19 e 30). Estes são tipos especiais, pois eles não são tipos primitivos ou tipos de classes. Na verdade eles são tipos de métodos. Para poder explicar melhor, eu anexei a este artigo a **Figura 2**. Nela você pode ver o Code Completion sugerindo que seja usado o evento OnClick de um TButton qualquer. O importante desta imagem está destacado por um retângulo. Veja que o evento OnClick dos botões é considerado pelo compilador como propriedades do tipo TNotifyEvent. Isto significa que assim como você pode criar classes para serem tipos de variáveis, também é possível criar tipos personalizados para serem considerados como métodos que poderão ser executados a qualquer momento.

E como você pode criar estes tipos especiais? Simples. Veja a declaração dos tipos TAfterSave e TAfterRemove respectivamente nas Linhas 05 e 06 da **Listagem 2**. Os tipos declarados nestas linhas são métodos que não possuem parâmetros, mas na **Listagem 4** coloquei um exemplo de como você pode declarar um tipo de procedimento ou função que deve receber um parâmetro para ser executado.

**Listagem 4**. Exemplos de como criar tipos de métodos

```
01 type
02  TAfterSave = procedure(CodigoDoRegistro: Integer);
03  TAfterRemove = function(CodigoDoRegistro: Integer): Boolean;
```

**Nota**

Não se preocupe com a diretiva stdcall nas Linhas 05 e 06 da **Listagem 2**. Ela serve apenas para garantir que os atributos dos métodos daquele tipo sejam passados na ordem correta para a DLL, mesmo que a DLL tenha sido criada em outra linguagem. Os tipos TAfterSave e TAfterRemove não possuem parâmetros, porém é uma boa ideia adicionar a diretiva stdcall quando se está fazendo chamada a métodos de uma DLL.

O adaptador ainda não está pronto. Ainda faltam as implementações dos métodos FreePlugin e GetDllName, além do construtor e destrutor. Você pode ver a declaração destes métodos na **Listagem 2**. Na **Listagem 5** se encontram as implementações dos mesmos.

**Listagem 5**. Métodos do adaptador TPluginAdapter

```
01 constructor TPluginAdapter.Create(DllFileName: string);
02 begin
03  inherited Create;
04  Self.DllFileName := DllFileName;
05  DllHandle := 0;
06 end;
07 procedure TPluginAdapter.FreePlugin;
08 begin
09  if DllHandle <> 0 then
10  begin
11    FreeLibrary(DllHandle);
12    DllHandle := 0;
13  end;
14 end;
15 destructor TPluginAdapter.Destroy;
16 begin
17  FreePlugin;
18  inherited;
19 end;
20 function TPluginAdapter.GetDllName: string;
21 begin
22  Result := DllFileName;
23 end;
24 end.
```

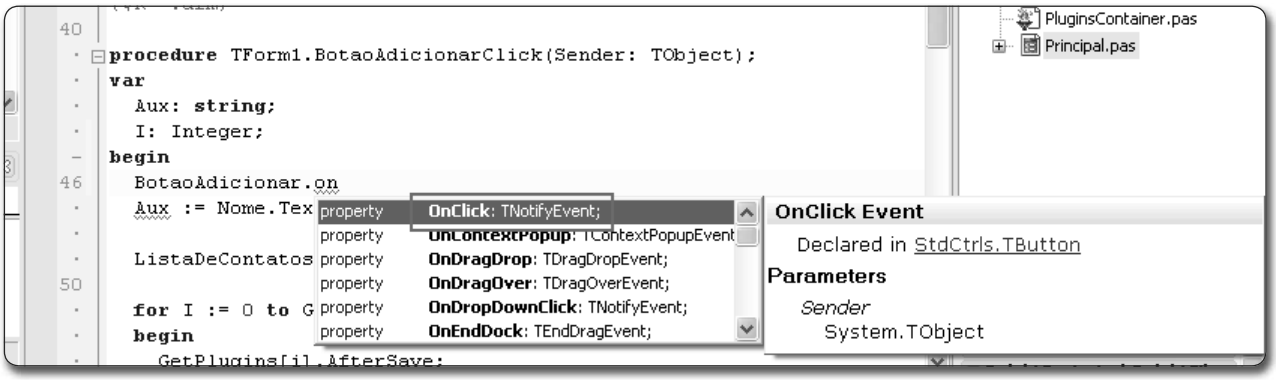



**Figura 2.** O tipo do evento OnClick

A **Listagem 5** apresenta a implementação dos métodos da classe TPluginAdapter que estavam faltando. Na Linha 1 está o construtor da classe. Para poder criar um objeto (uma instância), da classe TPluginAdapter é necessário informar o caminho de um arquivo DLL cujo adaptador irá encapsular as chamadas dos métodos AfterSave e AfterRemove. O caminho da DLL é passado como parâmetro do construtor e na Linha 4 você pode ver que o caminho é armazenado na variável DllFileName. A declaração desta variável está na Linha 10 da **Listagem 2**. Além disto, ainda no construtor da classe TPluginAdapter a variável DllHandle é iniciada (Linha 5 da **Listagem 5**).

A tradução de handle para o português é manipulador. Ela tem este nome, pois é usada para armazenar o valor (um número) que identifica o manipulador do arquivo DLL. De maneira resumida, este manipulador é como se fosse um ponteiro que indica o local da memória onde a DLL foi carregada. É através do valor desta variável que se torna possível fazer chamada aos métodos que estão na DLL.

Seguindo, na Linha 7 da **Listagem 5** está o método FreePlugin. Este método é usado pelo destrutor da classe, como você pode ver na Linha 17. É bem simples, pois ele apenas verifica se existe um manipulador criado para a DLL, o que significa que ela foi carregada para memória e caso o teste retorne verdadeiro a DLL é descarregada usando o método FreeLibrary (como pode ser visto na Linha 11). A última implementação é do método GetDllName. Este método serve apenas para retornar o valor que está dentro da variável DllFileName. Lembre-se que nesta variável está o caminho do arquivo DLL que a classe TPluginAdapter está adaptando.

## A última alteração

Com o adaptador TPluginAdapter pronto falta apenas implementar os dois últimos métodos (**Listagem 6**), muito simples, da classe TForm1. São os métodos do evento OnClick do botão BotaoProcurarDll e o método GetPlugins que é usado nas Linhas 13, 15, 24 e 26 da **Listagem 1**.

Observe a **Listagem 6**. Ela apresenta a implementação dos métodos GetPlugins e BotaoProcurarDllClick que faltavam na **Listagem 1**. O método GetPlugins tem o objetivo de criar e retornar a variável Plugins declarada na Linha 2. Já o método BotaoProcurarDllClick é responsável por procurar pelas DLL's no diretório onde o EXE está instalado. Na Linha 27, é armazenado na variável PastaDeDlls o caminho da pasta onde o EXE está instalado, e na Linha 28 é obtido um array de strings com o caminho de todas as DLL's encontradas no diretório do EXE. Na Linha 30 é feita uma iteração com o array de strings retornado pela função GetFiles e para cada DLL encontrada é criado um objeto da classe TPluginAdapter que será adicionado na lista de Plugins (Linhas 32 e 33).

**Listagem 6.** Outros métodos do fomulário de cadastro

```
01  private
02    Plugins: TList<TPluginAdapter>;
03    function GetPlugins: TList<TPluginAdapter>;
04  public
05    { Public declarations }
06  end;
07  var
08  Form1: TForm1;
09  implementation
10  uses Types, IOUtils;
11  {$R *.dfm}
12  { implementação do método BotaoAdicionarClick }
13  { implementação do método BotaoExcluirContatoClick }
14  function TForm1.GetPlugins: TList<TPluginAdapter>;
15  begin
16   if not Assigned(Plugins) then
17     Plugins := TList<TPluginAdapter>.Create;
18   Result := Plugins;
19  end;
20  procedure TForm1.BotaoProcurarDllClick(Sender: TObject);
21  var
22   PastaDeDlls: string;
23   DllsEncontradas: TStringDynArray;
24   I: Integer;
25   Plugin: TPluginAdapter;
26  begin
27   PastaDeDlls := TDirectory.GetParent(ParamStr(0));
28   DllsEncontradas := TDirectory.GetFiles(PastaDeDlls, '*.dll');
29   GetPlugins.Clear;
30   for I := Low(DllsEncontradas) to High(DllsEncontradas) do
31   begin
32     Plugin := TPluginAdapter.Create(DllsEncontradas[I]);
33     GetPlugins.Add(Plugin);
34   end;
35  end;
```

**Nota**

Lembre-se de declarar a variável Plugins como pode ser visto na Linha 2 da **Listagem 6**. Além disto, lembre-se também de declarar as units PluginAdapter e Generics.Collections na sessão uses da unit do formulário.

Neste ponto você já pode compilar o programa e ver que ele funciona mesmo sem Plugins. Observe que mesmo sem nenhum Plugin criado o programa não gera exceções.

## Criando um plugin para a aplicação

Já foi dito neste artigo que os Plugins nada mais são do que arquivos DLL que você, sua equipe ou qualquer outro programador do mundo crie. Essas DLL's podem ser escritas em qualquer linguagem e esta é uma das vantagens de se trabalhar com DLL's. Para que outros programadores criem Plugins para sua aplicação você precisa disponibilizar algum tipo de documentação descrevendo qual o objetivo do mesmo, quais são os métodos que deve ter e em que momento sua aplicação faz chamadas a estes métodos. Algo parecido com o que apresento a seguir:

**Documentação dos Plugins da Aplicação de Exemplo**

**Nome do sistema:** Aplicação de Exemplo

**Plugin 1**
**Objetivo:** Interagir com as funções de salvar e excluir registros.

**Métodos do plugin**
*procedure AfterSave();*
Este procedimento é invocado pela aplicação após um registro qualquer ser salvo.

*procedure AfterRemove();*
Este procedimento é invocado pela aplicação após a exclusão de um registro qualquer.

Com a documentação do Plugin em mãos e após entender como o Plugin funciona, é hora de criar o Plugin par interagir com a aplicação. Crie uma pasta em um local diferente do local onde você criou a aplicação de cadastro de contatos. Em seguida crie um novo projeto de DLL no Delphi 2010 acessando o menu File>New>Other e escolhendo a opção Dynamic-link Library. Salve o novo projeto criado com o nome PluginMostraMensagem dentro da nova pasta que você acabou de criar. Veja o código desta DLL de Plugin na **Listagem 7**.

**Listagem 7**. Um plugin para exibir mensagens

```
01 library PluginMostraMensagem;
02 uses Dialogs;
03 {$R *.res}
04 procedure AfterSave; stdcall;
05 begin
06  ShowMessage('Um registro foi salvo');
07 end;
08 procedure AfterRemove; stdcall;
09 begin
10  ShowMessage('Um registro foi excluído');
11 end;
12 exports
13  AfterSave, AfterRemove;
14 end.
```

A **Listagem 7** mostra um Plugin que tem o objetivo de mostrar mensagens sempre que um registro for salvo ou excluído. O objetivo do Plugin varia de acordo com cada programador e depende apenas da criatividade de cada um. A única regra é que o Plugin respeite as regras definidas na documentação para que funcione corretamente.

Ao compilar este novo projeto será criada uma DLL dentro da pasta onde você salvou o projeto. Não adianta pressionar F9, pois projetos DLL não podem ser executados. Eles só podem ser compilados. Você deve usar a tecla de atalho CTRL + F9 para projetos de DLL.

## Instalando o novo plugin

Antes de copiar o Plugin para a pasta do EXE da aplicação, faça o seguinte teste. Abra a aplicação e clique no botão "Procurar e registrar os plugins encontrados". Isto fará com que a aplicação procure por todas as DLL's existentes na pasta do EXE e é claro que nenhum arquivo será encontrado. Após, adicione e remova alguns contatos para ver se algum erro é gerado. Em seguida, sem fechar a aplicação, copie a DLL que você criou para a pasta do EXE. Clique no botão "Procurar e registrar os plugins encontrados" novamente. Isto fará com que a aplicação encontre sua nova DLL de plugin. Por último, adicione e remova alguns contatos e veja se o plugin vai interagir com aplicação exibindo as mensagens 'Um registro foi salvo' e 'Um registro foi excluído' no momento adequado.

Você percebeu que o programa começou a mostrar mensagens após salvar ou excluir um registro sem precisar ser recompilado. Nem ao menos foi necessário fechá-lo. Tudo que você precisou fazer foi copiar a DLL para a pasta do EXE e pedir para a aplicação procurar por ela. A partir daí o programa passou a ter outro comportamento para os cadastros.

Não se limite apenas a um Plugin. Da maneira como foi feita a arquitetura deste exemplo, você pode criar mais DLL's de Plugins e a aplicação se responsabiliza por interagir com cada DLL colocada no diretório do EXE. Repare também que mesmo que a DLL não possua os métodos corretos, nenhuma exceção será gerada.

## Plus

Você não precisa ler o restante deste artigo para entender como funciona um sistema orientado a Plugins, pois já viu tudo que precisa saber para criar um programa com esta arquitetura. Porém, se você pensar em termos de manutenção verá que o sistema está um tanto rígido, pois todo o código de procura e registro dos Plugins ficou espalhado pela aplicação. Isto será terrível a curto e médio prazo. Veja nas **Listagens 1** e **6** que a classe TForm1 possui toda a lógica de negócio para procurar uma DLL de Plugin e criar um objeto TPluginAdapter para a mesma. Isto não é bom, pois você terá que duplicar este código por todos os lugares da aplicação que precisar usar Plugins. Por sorte, existem alternativas que podem ajudar a ter uma arquitetura de Plugins sem precisar duplicar tantas linhas de códigos ou fazer muitas referências entre as units da aplicação. A seguir vou apresentar uma destas alternativas.

A ideia é bem simples. Existirá uma classe chamada TPluginsContainer e ela será a responsável por interagir com todas as DLL de Plugin que possam existir. Desta maneira vou retirar a lógica de negócios referente aos Plugins da classe TForm1 e de todas as demais classes que usem os Plugins para colocar dentro da nova classe chamada TPluginsContainer. Veja a primeira parte do código da classe TPluginsContainer na **Listagem 8**.

A **Listagem 8** mostra o 'cabeçalho' de uma classe um tanto incomum. Ela é diferente, pois não expõe o seu construtor para as demais classes do sistema. Veja na Linha 09 que o

construtor está declarado com nível de acesso protegido, o que significa que somente as subclasses de TPluginsContainer ou ela própria poderá usar este método. Em outras palavras, nenhuma outra classe será capaz de criar um objeto da classe TPluginsContainer usando a instrução TPluginsContainer.Create. Então, como as demais classes poderão usar esta classe se não é possível criar uma instância dela? As demais classes podem conseguir uma instância desta classe através do método GetInstance declarado na Linha 11. E para que serve isto? Este é um padrão conhecido como Singleton. Este padrão garante que somente uma instância da mesma classe exista no sistema inteiro. Em outras palavras é impossível criar dois objetos diferentes para a classe TPluginsContainer. Pense comigo. Se esta classe serve apenas para procurar Plugins e criar os adaptadores para cada um deles, não é necessário que exista mais de um objeto desta classe ao mesmo tempo. Qual seria a vantagem de ter dois objetos da classe TPluginsContainer procurando pelos Plugins no mesmo lugar? Tornar o container de Plugins uma classe Singleton faz todo sentido, pois levando em consideração que esta classe serve apenas para localizar DLL's e criar os adaptadores para cada DLL encontrada, não seria vantagem alguma existir mais de uma instância da classe TPluginsContainer, sendo assim um desperdício de recursos. Observe também que o método GetInstance é um método de classe, pois possui a palavra reservada class no início da declaração. Isto significa que é possível chamar este método diretamente da classe, como você verá mais adiante.

**Listagem 8**. O cabeçalho do container

```
01 unit PluginsContainer;
02 interface
03 uses Generics.Collections, PluginAdapter;
04 type
05  TPluginsContainer = class
06  private
07    ListaDePlugins: TList<TPluginAdapter>;
08  protected
09    constructor Create;
10  public
11    class function GetInstance: TPluginsContainer;
12    procedure ProcurarDlls;
13    procedure AfterSave;
14    procedure AfterRemove;
15  end;
16 implementation
17 { Implementação dos métodos aqui }
```

Em seguida, na Linha 12 da **Listagem 8**, existe a declaração do método ProcurarDlls que fará a busca pelas DLL's. Na **Listagem 9** você verá que a implementação deste método é igual à implementação do BotaoProcurarDllClick da **Listagem 6**. Isto porque estou extraindo da classe TForm1 todo o código que diz respeito à manipulação de Plugins para juntá-los em uma única classe, evitando assim a duplicidade de código entre as classes que precisam acessar os Plugins.

## A implementação do container

A seguir note a implementação da classe Singleton, apresentada na **Listagem 9**.

Na **Listagem 9** você pode ver como o padrão Singleton funciona e como é possível garantir que exista apenas um objeto container em toda a aplicação. Primeiro é necessário ter em mente que em nenhum outro lugar será possível criar objetos com a instrução TPluginsContainer.Create, ao menos que seja dentro da própria classe TPluginsContainer, pois o construtor foi declarado como protegido. As outras classes precisarão chamar o método GetInstance para conseguir um objeto desta classe. Veja entre as Linhas 12 e 17 que o método GetInstance verifica se já existe um objeto criado na variável Instance, declarada na Linha 04. Caso ainda não exista um objeto, o único é criado nesta variável, e caso já exista o método apenas retorna a variável como resultado da função. Em outras palavras, o método GetInstance é o único método a ser usado para conseguir um objeto da classe TPluginsContainer e internamente este método sempre retorna o mesmo objeto, como mostrado na Linha 16. É desta maneira que o padrão Singleton garante a existência de um único objeto da mesma classe para toda a aplicação. A **Listagem 10** mostra a implementação do restante da classe.

**Listagem 9**. O código de uma classe Singleton

```
01 implementation

02 uses IOUtils, Types;

03 var
04  Instance: TPluginsContainer;

05 {TPluginsContainer }

06 constructor TPluginsContainer.Create;
07 begin
08  inherited;
09  ListaDePlugins := TList<TPluginAdapter>.Create;
10  ProcurarDlls;
11 end;

12 class function TPluginsContainer.GetInstance: TPluginsContainer;
13 begin
14  if not Assigned(Instance) then
15  Instance := TPluginsContainer.Create;
16  Result := Instance;
17 end;
```

**Listagem 10**. A implementação do container

```
01 procedure TPluginsContainer.ProcurarDlls;
02 var
03  PastaDeDlls: string;
04  DllsEncontradas: TStringDynArray;
05  I: Integer;
06  Plugin: TPluginAdapter;
07 begin
08  PastaDeDlls := TDirectory.GetParent(ParamStr(0));
09  DllsEncontradas := TDirectory.GetFiles(PastaDeDlls, '*.dll');
10  ListaDePlugins.Clear;
11  for I := Low(DllsEncontradas) to High(DllsEncontradas) do
12  begin
13    Plugin := TPluginAdapter.Create(DllsEncontradas[I]);
14    ListaDePlugins.Add(Plugin);
15  end;
16 end;

17 procedure TPluginsContainer.AfterRemove;
18 var
19  I: Integer;
20 begin
21  for I := 0 to ListaDePlugins.Count - 1 do
22    ListaDePlugins[I].AfterRemove;
23 end;

24 procedure TPluginsContainer.AfterSave;
25 var
26  I: Integer;
27 begin
28  for I := 0 to ListaDePlugins.Count - 1 do
29    ListaDePlugins[i].AfterSave;
30 end;
```

Você vai perceber uma grande semelhança das **Listagens 1** e **6** com a **Listagem 10**. Se você olhar com mais atenção verá que o método ProcurarDlls da **Listagem 10** é o mesmo método do BotaoProcurarDllClick da **Listagem 6** e além disto o código dos métodos AfterRemove e AfterSave da **Listagem 10** corresponde respectivamente ao código das Linhas 24 a 27 e 13 a 16 da **Listagem 1**. Isto acontece, porque estou removendo da classe TForm1 todo código que diz respeito à procura e registro dos Plugins. Ao remover este código da classe TForm1 eu centralizei toda a lógica de negócio referente aos Plugins em uma única classe, o que facilita a manutenção do sistema tornando-o muito mais flexível. Veja na **Listagem 11** como a classe TForm1 ficou mais simples após a criação do container de Plugins.

**Listagem 11**. A versão final da classe TForm1

```
01 implementation
02 uses PluginsContainer;
03 {$R *.dfm}
04 procedure TForm1.BotaoAdicionarClick(Sender: TObject);
05 var
06  Aux: string;
07 begin
08  Aux := Nome.Text + ' - ' + Endereco.Text + ' - ' + Cep.Text;
09  ListaDeContatos.Items.Add(Aux);
10  TPluginsContainer.GetInstance.AfterSave;
11 end;
12 procedure TForm1.BotaoExcluirContatoClick(Sender: TObject);
13 begin
14  if ListaDeContatos.Count > 0 then
15    ListaDeContatos.DeleteSelected;
16  TPluginsContainer.GetInstance.AfterRemove;
17 end;
18 procedure TForm1.BotaoProcurarDllClick(Sender: TObject);
19 begin
20  TPluginsContainer.GetInstance.ProcurarDlls;
21 end;
```

**Nota**

Não se esqueça de alterar a sessão uses da classe TForm1 para que fique como é apresentado na Linha 2 da **Listagem 11**. Além disto remova a declaração do método GetPlugins e da variável Plugins que está na sessão private da classe TForm1.

Você pode notar na **Listagem 11** que os métodos GetPlugins, BotaoProcurarDllClick sumiram da classe TForm1. Isto porque foram colocados dentro da classe TPluginsContainer. Desta maneira, qualquer outra classe pode usar os Plugins da aplicação sem a necessidade de conhecer a regra de negócios sobre os Plugins. Veja nas Linhas 10, 16 e 20 da **Listagem 11** que para conseguir uma instância da classe TPluginsContainer é necessário chamar o método GetInstance, uma vez que o construtor da classe está escondido.

## Conclusão

Você viu neste artigo que sistemas orientados a Plugins possuem uma arquitetura simples de criar e não exigem grande experiência do programador com a linguagem de desenvolvimento. Sistemas com uma arquitetura orientada a Plugins facilitam a manutenção do sistema, pois separam o projeto em pequenos módulos que podem sofrer alterações independentes do restante da aplicação, além de permitir que outros programadores possam criar novas funções para o seu programa. Aconselho que você baixe o exemplo do artigo e tente criar diferentes tipos de Plugins para o seu programa. Pense como seria possível criar um Plugin que exiba uma pergunta de confirmação antes de excluir um registro? Para isto será necessário criar mais um método para a arquitetura existente. Um novo método que talvez possa se chamar BeforeRemove. Este método na verdade será uma função e que internamente (na DLL), faz uma pergunta ao usuário e retorna True se o usuário confirmar a exclusão do registro. Agora é sua vez. Baixe o exemplo disponível e boa diversão. ●

### Nota do DevMan

O padrão Singleton cria um ponto comum de acesso para um objeto de uma classe, escondendo o construtor da classe que se deseja que exista apenas um ÚNICO OBJETO. Ao esconder o construtor da classe se torna impossível criar um objeto da maneira convencional que seria TNomeDaClasse.Create. Ao invés disto uma classe Singleton deve disponibilizar um método de classe, normalmente chamado de GetInstance, para obter um OBJETO ÚNICO da classe que é usado por toda a aplicação. Uma das vantagens da utilização deste padrão é o baixo uso de recursos de hardware, pois garante que não existirá mais de um objeto para uma mesma classe, poupando o uso desnecessário de memória.

**Links**

**Desvendando Generics e Métodos anônimos no Delphi 2009**
*http://cc.embarcadero.com/Item/26347*

**Plugin**
*http://pt.wikipedia.org/wiki/Plugin*

**Creating and Using DLLs from Delphi**
*http://delphi.about.com/od/windowsshellapi/a/dll_basics.htm*

**Creating a real singleton class in Delphi**
*http://edn.embarcadero.com/article/22576*

**Singleton pattern**
*http://www.castle-cadenza.demon.co.uk/single.htm*

**Singleton**
*http://pt.wikipedia.org/wiki/Singleton*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

# Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

# Dicas de Delphi for PHP

## Google Maps, Segurança, Banco de Dados e JavaScript

**Do que trata o artigo**

Trataremos neste artigo algumas dicas com a ferramenta Delphi for PHP. O principal objetivo é mostrar de forma clara e objetiva a solução de problemas que ocorrem em determinados projetos do nosso dia-a-dia, como a necessidade de usar código JavaScript e garantir a segurança da aplicação.

**Para que serve**

As técnicas aqui apresentadas podem ser aplicadas nos mais diferentes tipos de projetos Web com PHP. Por exemplo, o uso de JavaScript pode incrementar a experiência do usuário incluindo funcionalidade client (que roda no browser), sem que precise ser feita uma requisição ao servidor. A integração com o Google Maps pode trazer uma série de benefícios ao seu site, por exemplo, a capacidade de exibir detalhadamente a localização de clientes, a partir de seus endereços.

**Em que situação o tema é útil**

Ao desenvolver sites PHP, você precisará utilizar muitas das dicas apresentadas neste artigo. Cada recurso tem sua melhor utilidade em cada situação, Mapas do Google são úteis em diversos casos, desde a indicação da localização de estabelecimentos, até como plataforma para criação de serviços de rastreamento. O DBGrid é a forma mais útil e fácil de exibir dados em formulários Web, como veremos aqui. Com o advento do AJAX, conhecer e usar JavaScript tornou-se um diferencial em qualquer aplicação Web. Autenticar usuários é tarefa básica de qualquer site, e veremos que com o Delphi for PHP, essa tarefa se torna bastante simples.

**Rafael Silva**
*rafael@rlsystem.com.br*
*Trabalha com análise e desenvolvimento de sistemas. Atua na área de desenvolvimento desktop utilizando Delphi, C#, e foco na web com PHP, ASP.NET, CSS, AJAX, sempre utilizando os padrões da Web 2.0 em seus projetos. Utiliza banco de dados MySQL, SQL Server, PostGreSQL e Firebird. Para outras informações acesse o site, http://www.rlsystem.com.br. Palestras e treinamentos entre em contato pelo email.*

## Resumo do DevMan

Programadores PHP sempre foram acostumados a programar tudo no "braço", apesar de existirem IDE's e ferramentas como o ZEND e o Eclipse que facilitam esse processo. Normalmente desenvolver sites com PHP consiste em escrever código para todas as funcionalidades da página. Em contrapartida, ao desenvolver com Delphi for PHP, temos o suporte de um IDE onde podemos utilizar componentes e desenvolvimento visual, configurar propriedades, utilizar um designer e muito mais. O lançamento do Delphi for PHP foi uma grande evolução. Um dos objetivos deste artigo é mostrar situações comuns no desenvolvimento com PHP básico, porém apresentando como é bem mais simples desenvolver agora na ferramenta Delphi For PHP.

A maioria dos programadores PHP sempre realizou todas suas tarefas no "braço". Qualquer projeto, por menor que fosse, já nos gerava um certo trabalho. Apesar disso, o PHP sempre foi a linguagem favorita para o desenvolvimento Web, principalmente se a compararmos com o ASP clássico. O PHP apresenta uma maior produtividade e até mesmo performance. Apesar de termos alguns IDE's para PHP, sem dúvida o Delphi for PHP é um dos que apresenta os melhores recursos de desenvolvimento visual, além de um excelente framework. Com isso, temos um IDE produtivo, aliado a uma excelente linguagem de desenvolvimento Web.

Neste artigo apresento soluções para as principais situações que ocorriam em meu dia-dia de desenvolvimento de web sites com PHP. Além disso, coloquei aqui algumas soluções para dúvidas comuns que costumo receber por email. Comprovaremos que desenvolver aplicações para Web com Delphi for PHP é tão simples e intuitivo quanto o desenvolvimento de aplicações Win32 com VCL.

## Criando os projetos demonstrados neste artigo

Antes de iniciarmos é importante você entender como proceder para criar um projeto no Delphi for PHP. Para aqueles que já possuem conhecimento da ferramenta, sugiro já ir avançando nos tópicos. Partindo do princípio que você já tenha realizado o download e a instalação do Delphi For PHP (veja sessão links), execute o mesmo, clique em Arquivo>Novo>Aplicação e em seguida Arquivo>Salvar Todos. Escolha um diretório de sua preferência e altere o nome de sua unit para index.php. Logo após salve seu projeto. A cada dica será criado um novo projeto.

**Nota**

O PHP trabalha com o servidor Apache, por padrão a maioria dos servidores (Apache) são configurados para sempre procurar o arquivo com nomenclatura "index", seja index.html ou index.php, por isso é de extrema importância que você saiba que o nome de sua página inicial deverá se chamar index.php.

## Banco de dados

Vamos iniciar criando o banco de dados que será utilizado neste artigo. Partindo do ponto que você já possua um servidor MySQL instalado (veja sessão links), crie um novo banco de dados com o nome de sua preferência e em seguida adicione uma nova tabela (denominada clientes) com os campos ID, NOME, CIDADE e ESTADO. Confira na **Listagem 1** o script SQL da criação da tabela e a criação de alguns registros usando INSERT.

**Nota**

Aqui usaremos o MySQL, mas vale lembrar que o Delphi for PHP suporta outros bancos de forma nativa, como o PostgreSQL, Oracle, SQL Server e Interbase.

## Listando dados em um DBGrid

Você que já programa em PHP tente voltar ao nosso cenário "clássico". Lembra quando tínhamos que exibir uma lista de produtos, por exemplo, na parte do site responsável pela administração da aplicação? Criávamos uma conexão com o banco de dados, definíamos uma tabela HTML em código e também usando código realizávamos um loop em uma tabela até ser gerada uma listagem de dados. No Delphi for PHP você simplesmente arrasta um componente Database (para realizar a conexão com banco de dados), um Table (para acessar uma tabela) e um Datasource (para a ligação entre nossa tabela e nosso controle de exibição de dados). Então faz a ligação entre os componentes e exibe dados em controles de tela, sem necessidade de código algum. Este será nosso primeiro exemplo.

**Listagem 1**. Script SQL

```
CREATE TABLE `artigo`.`clientes` (
`ID` INT( 11 ) NOT NULL AUTO_INCREMENT PRIMARY KEY,
`NOME` VARCHAR( 100 ) NOT NULL ,
`CIDADE` VARCHAR( 100 ) NOT NULL ,
`ESTADO` VARCHAR( 100 ) NOT NULL
) ENGINE = MYISAM ;

INSERT INTO `clientes` (`ID`, `NOME`, `CIDADE`, `ESTADO`) VALUES
(1, 'Rafael', 'Guaíba', 'RS'),
(2, 'João', 'São Paulo', 'SP'),
(3, 'Maria', 'Rio de Janeiro', 'RJ'),
(4, 'Amanda', 'Curitiba', 'PR'),
(5, 'Pedro', 'Torres', 'RS'),
(6, 'Souza', 'Gramado', 'RS'),
(7, 'Paula', 'Florianópolis', 'SC'),
(8, 'Carol', 'Gravataí', 'RS');
```

Adicione então ao formulário os componentes Database, Datasource e Table (todos da paleta Data Access). Altere suas propriedades Name para Banco, DSClientes e TBClientes, respectivamente. Configure as propriedades dos componentes conforme a **Tabela 1**.

| **Banco** | |
|---|---|
| DatabaseName | Artigo |
| UserName | Root |
| UserPassword | 123 |
| Host | localhost |
| Connected | True |
| **DSClientes** | |
| DataSet | TBClientes |
| **TBClientes** | |
| Database | Banco |
| TableName | Clientes |
| Active | True |

**Tabela 1**. Componentes e propriedades

Em seguida adicione um DBGrid e altere sua propriedade DataSource apontando para o DSClientes. Observe na **Figura 1** o resultado do nosso exemplo.

## Google Maps

A tecnologia Google Maps com certeza é um dos melhores novos recursos da web 2.0, é muito comum encontrar em qualquer site empresarial, por exemplo, um mapa utilizando a tecnologia Google Maps para exibir "como chegar na empresa". Integrar o mesmo com o PHP não é uma tarefa tão complexa,

mas também não tão simples, é preciso trabalhar um pouco com JavaScript e alguns códigos PHP. No Delphi For PHP, temos disponível um componente que centraliza todo esse acesso para nós programadores (GoogleMap, paleta Google), onde simplesmente você define a propriedade Address e em tempo de execução o mesmo já é atualizado. Crie uma nova aplicação e salve a mesma como GoogleMaps. Na unit principal adicione um componente GoogleMap, altere sua propriedade Name para GoogleMaps. Em seguida, adicione um Edit e um Button, modificando a propriedade Name do Edit para EdtEndereco e as propriedades Name e Caption do Button, como btnOK e OK respectivamente. Dê um duplo clique no Button e adicione o código a seguir no seu evento OnClick:

```
$this->GoogleMaps->Address = $this->EdtEndereco->Text;
```

**Figura 1.** Resultado - DBGrid

Note que simplesmente definimos sua propriedade Address com o valor que será digitado no Edit. Na **Listagem 2** você confere o código completo. Note que o Delphi for PHP gera de forma automática praticamente todo o código da listagem, como em uma aplicação Delphi Win32. O único código que foi necessário inserir foi o que vincula o endereço digitado no Edit ao endereço do componente GoogleMaps. Note na **Listagem 2** a similaridade com uma classe de um formulário Delphi (coloquei comentários para facilitar o entendimento do código que é gerado). Nela temos as cláusulas de importação de arquivos (units), a definição da classe da página (o form) e seus membros. Veja o resultado deste exemplo na **Figura 2**.

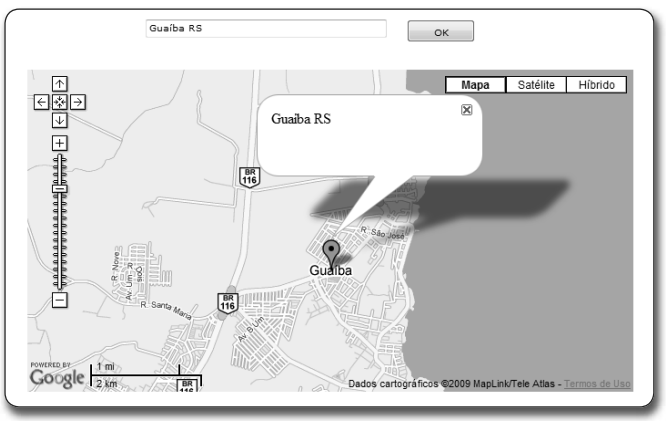



**Figura 2.** Resultado - Google Maps

## Autenticação básica

Criar um sistema de autenticação não é uma tarefa simples, mas exige conhecimento. Digo isso simplesmente pelos possíveis ataques a que o nosso site possa estar vulnerável. Particularmente já encontrei muitos sites com vulnerabilidade de SQL Injection e é sempre preciso tomar muito cuidado ao desenvolver um sistema de login, pois é onde a maioria dos hackers costuma atacar. O procedimento consiste em escrever código SQL no local onde são esperados o login e senha do usuário. O servidor recebe então essas instruções e as executa, ao invés de usá-las como parâmetros em cláusulas where. Esse é um exemplo típico de vulnerabilidade.

**Listagem 2**. Código completo da unit principal (projeto GoogleMaps)

```
<?php
require_once("vcl/vcl.inc.php");
//Inclusões
use_unit("google/maps/gmaps.inc.php");
use_unit("forms.inc.php");
use_unit("extctrls.inc.php");
use_unit("stdctrls.inc.php");

//Definição de classe
class Unit1 extends Page
{
       public $GoogleMaps = null;
       public $btnOK = null;
       public $EdtEndereco = null;
       function btnOKClick($sender, $params)
       {
          $this->GoogleMaps->Address = $this->EdtEndereco->Text;
       }
}

global $application;

global $Unit1;

//Cria o formulário
$Unit1=new Unit1($application);

//Ler do arquivo de recursos
$Unit1->loadResource(__FILE__);

//Mostrar o formulário
$Unit1->show();

?>
```

### Nota do DevMan

O Google Maps é um serviço de pesquisa e visualização de mapas e imagens de satélite da Terra disponível na Web, fornecido e desenvolvido pela empresa Google. Atualmente, o serviço disponibiliza mapas e rotas para qualquer ponto nos Estados Unidos, Canadá, na Europa, Austrália, Brasil, entre outros. Também permite a visualização de imagens via satélite do mundo todo, com possibilidade de zoom nas grandes cidades, como Nova York, Paris e São Paulo. Juntamente com o lançamento da versão brasileira do Google Maps, a empresa introduziu o Local Business Center, ferramenta que permite que qualquer empresa faça seu cadastro e seja então encontrada no Google Maps por qualquer usuário. No cadastro as empresas podem preencher dados como horário de atendimento, formas de pagamento, logotipo e fotos, sendo necessária confirmação do cadastro através de uma ligação telefônica, SMS ou carta. O cadastro no Local Businesse Center, apesar de ser um procedimento extremamente simples, foge ao escopo deste artigo, para mais informações veja sessão Links.

Já desenvolvi muitos sites com esse requisito, em sua maioria, com rotinas criadas manualmente, utilizando autenticação via HTTP (do servidor Apache). No Delphi For PHP temos um componente simples que cuida de todo o trabalho "pesado". Para este exemplo, crie um novo projeto (BasicAuthentication) e em seguida adicione ao formulário um componente BasicAuthentication da paleta System. Primeiramente altere seu nome para "autentica" e defina suas propriedades ErrorMessage, Password e Username. As próprias nomenclaturas já falam por si o que cada uma delas deve conter. Com essas informações já definidas, selecione o formulário, clique em Eventos, no Object Inspector escolha o evento OnBeforeShow e digite o seguinte código (para autenticar o usuário antes da exibição da página):

```
$this->autentica->Execute();
```

Na **Listagem 3** você confere o código completo. Veja o resultado conforme mostra a **Figura 3**.

**Listagem 3**. Código completo da unit principal (projeto BasicAuthentication)

```
<?php
require_once("vcl/vcl.inc.php");
//Inclusões
use_unit("auth.inc.php");
use_unit("chart.inc.php");
use_unit("forms.inc.php");
use_unit("extctrls.inc.php");
use_unit("stdctrls.inc.php");

//Definição de classe
class Unit1 extends Page
{
        public $Button1 = null;
        public $autentica = null;
        function Unit1BeforeShow($sender, $params)
        {
          $this->autentica->Execute();
        }
}

global $application;

global $Unit1;

//Cria o formulário
$Unit1=new Unit1($application);

//Ler do arquivo de recursos
$Unit1->loadResource(__FILE__);

//Mostrar o formulário
$Unit1->show();

?>
```

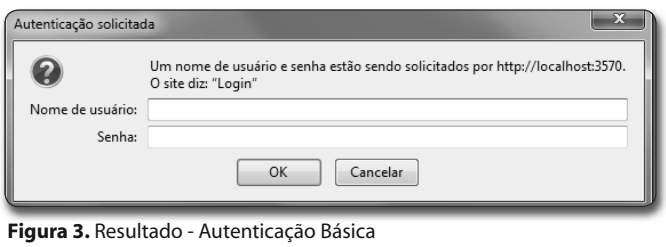



**Figura 3.** Resultado - Autenticação Básica

## DateTimePicker

A entrada de valores do tipo data é muito comum em aplicativos Web. Um exemplo é a obtenção de datas para geração de relatórios, como vendas por mês, por intervalo etc. Para essa tarefa, normalmente era criada uma espécie de máscara que formatava o texto (via JavaScript puro) conforme o usuário final fosse digitando a data em uma caixa de entrada. No Delphi For PHP não precisamos realizar trabalho extra, temos disponível o componente DateTimePicker.

Para vermos um exemplo de utilização deste componente, primeiramente crie um novo projeto e adicione um DateTimePicker da paleta Advanced. Configure sua propriedade IfFormat para %d/%m/%Y (ou outro formato que desejar) para que seja exibida na forma dia/mês/ano. Em seguida adicione um Label e Button (ambos da paleta Standard). Altere as propriedades Caption e Name do Button para Visualizar e BtnVisualizar respectivamente e as mesmas propriedades do Label para Resultado e lblResultado. Adicione ao evento OnClick do botão o código que segue:

```
$this->lblResultado->Caption = $this->DateTimePicker1->Text;
```

Veja que basicamente definimos a propriedade Text do Label com o valor recebido da propriedade Text do DateTimePicker. Você pode utilizar essa dica não só na entrada de datas para geração de relatórios, mas em cadastros (por exemplo, para informar a data de nascimento do usuário) e em muitas outras situações. Na **Listagem 4** você confere o código completo do projeto. O resultado final é exibido conforme ilustra a **Figura 4**.

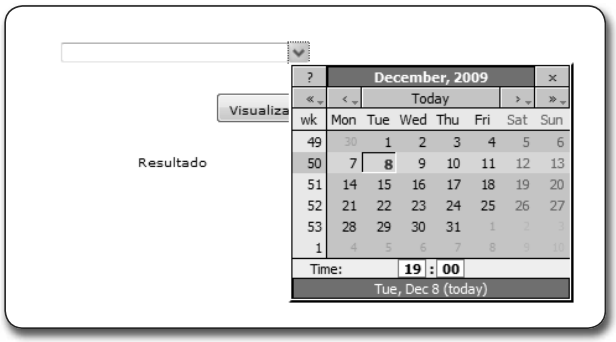



**Figura 4.** Resultado - DateTimePicker

## Upload de arquivos

Envio de arquivos para o servidor é um requisito de todo sistema que se julgue completo. Dificilmente seu cliente não

### Nota do DevMan

O servidor Apache, ou Apache HTTP Server, é o mais bem sucedido servidor web livre. Foi criado em 1995 por Rob McCool, até então, funcionário do NCSA (National Center for Supercomputing Applications). Numa pesquisa realizada em Dezembro de 2007, foi constatado que a utilização do Apache representa 47.20% dos servidores ativos no mundo. É a principal tecnologia da Apache Software Foundation, responsável por mais de uma dezena de projetos envolvendo tecnologias de transmissão via web, processamento de dados e execução de aplicativos distribuídos. O servidor é compatível com o protocolo HTTP versão 1.1. Suas funcionalidades são mantidas através de uma estrutura de módulos, permitindo inclusive que o usuário escreva seus próprios módulos utilizando a API do software. É disponibilizado em versões para os sistemas Windows, Novell Netware, OS/2, Unix, Linux, FreeBSD etc.

vai querer possuir um álbum de fotos, inserir imagens e/ou notícias de seu site entre outras situações. Desde que a Web 2.0 surgiu, esse tipo de interação com o visitante aumentou bastante. Veja por exemplo famosos sites como YouTube, Orkut entre diversos outros. Sem dúvida trabalhar com Upload de arquivos com PHP não é uma tarefa complicada, simplesmente é preciso adicionar um Input do tipo File e definir no Form da página o famoso enctype="multipart/form-data", para então tratar o recebimento do mesmo com "moved_upload_file" para envio de arquivos ao servidor. No Delphi For PHP já não precisamos definir essas informações no formulário, você tem disponível o componente Upload da paleta Additional.

**Listagem 4**. Código completo da unit principal (projeto DateTimePicker)

```
<?php
require_once("vcl/vcl.inc.php");
//Inclusões
use_unit("comctrls.inc.php");
use_unit("forms.inc.php");
use_unit("extctrls.inc.php");
use_unit("stdctrls.inc.php");

//Definição de classe
class Unit1 extends Page
{
       public $lblResultado = null;
       public $BtnVisualizar = null;
       public $DateTimePicker1 = null;
       function BtnVisualizarClick($sender, $params)
       {
         $this->lblResultado->Caption = $this->DateTimePicker1->Text;
       }

}

global $application;

global $Unit1;

//Cria o formulário
$Unit1=new Unit1($application);

//Ler do arquivo de recursos
$Unit1->loadResource(__FILE__);

//Mostrar o formulário
$Unit1->show();

?>
```

Para este exemplo crie um novo projeto chamado Upload e em seguida insira um Button e Upload. Nas propriedades do Button, defina seu Caption como Enviar Arquivo e Name como btnEnviar, no evento OnClick do mesmo insira o código seguir:

```
move_uploaded_file($_FILES["Upload1"]["tmp_name"],"C:\\".$_
FILES["Upload1"]["name"]);
```

Para recuperar este arquivo, utilizamos a função moved_uploaded_file, que necessita de dois parâmetros do tipo string: o primeiro é o nome do arquivo, com isso, utilizamos tmp_name que recupera o nome temporário do mesmo e no segundo parâmetro é informado o local onde será salvo. Na **Listagem 5** você confere o código completo. Veja o resultado conforme mostra a **Figura 5**.

## Cookies

Todos sabem o que são Cookies. Acredito que o uso mais comum dele seja para sistemas de Login, onde o usuário pode sair e entrar no site novamente, sem precisar realizar a autenticação novamente.

**Listagem 5**. Código completo da unit principal (projeto Upload)

```
<?php
require_once("vcl/vcl.inc.php");
//Inclusões
use_unit("Zend/zmail.inc.php");
use_unit("clock.inc.php");
use_unit("forms.inc.php");
use_unit("extctrls.inc.php");
use_unit("stdctrls.inc.php");

//Definição de classe
class Unit1 extends Page
{
       public $BtnEnviar = null;
       public $Upload1 = null;
       function BtnEnviarClick($sender, $params)
       {
         move_uploaded_file($_FILES["Upload1"]["tmp_name"],
          "C:\\".$_FILES["Upload1"]["name"]);
       }
}

global $application;

global $Unit1;

//Cria o formulário
$Unit1=new Unit1($application);

//Ler do arquivo de recursos
$Unit1->loadResource(__FILE__);

//Mostrar o formulário
$Unit1->show();

?>
```

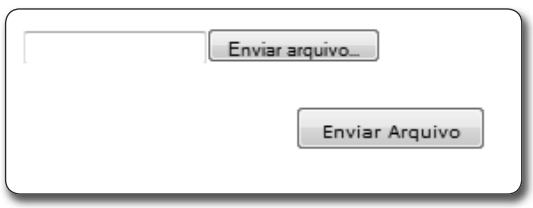



**Figura 5.** Resultado - Upload de arquivos

Para este exemplo crie uma nova aplicação denominada Cookies e em seguida adicione um Label, dois Button's e um Edit. Nas propriedades do Label defina seu Caption como vazio, Fonte Size 14px e Name para lblResultado, no primeiro Button altere as propriedades Caption e Name para Criar Cookie e BtnCriar respectivamente e no segundo para Visualizar Cookie e BtnRecuperar e por fim, defina a propriedade Name do Edit para EdtValor. A ideia desse exemplo é gravar determinada informação em um cookie para posterior recuperação. Então no evento OnClick do BtnCriar adicione o código a seguir:

```
setcookie("MyCookie", $this->EdtValor->Text, time()+3600);
```

Observe que simplesmente utilizamos a função setcookie, definindo os parâmetros nome do Cookie, valor do Cookie e o tempo de duração do mesmo, que neste caso equivale a uma hora. O valor do Cookie é recuperado através do valor digitado no Edit. Também no evento OnClick, porém do botão BtnRecuperar, adicione o código a seguir:

```
$this->lblResultado->Caption = $_COOKIE["MyCookie"];
```

## Nota do DevMan

Web 2.0 é um termo criado em 2004 pela empresa O'Reilly Media para designar uma segunda geração de comunidades e serviços, tendo como conceito a Web como plataforma, envolvendo Wiki's, redes sociais e Tecnologia da Informação. Embora o termo tenha uma conotação de uma nova versão para a Web, ele não se refere à atualização nas suas especificações técnicas, mas a uma mudança na forma como ela é encarada por usuários e desenvolvedores, ou seja, o ambiente de interação que hoje engloba inúmeras linguagens e motivações. Os exemplos de aplicações mais comuns da Web 2.0 são as redes sociais, como Facebook, Twitter, Linkedin, Orkut, YouTube etc.

Veja que o valor do Cookie é recuperado utilizando a variável global $_COOKIE informando como parâmetro o nome do mesmo e atribuindo o resultado à propriedade Caption do Label. Com esta dica, você está apto a criar um sistema de Login ou qualquer outra aplicação para posterior recuperação de dados.

**Nota**

Você precisa estar ciente que é preciso criptografar determinadas informações quando se trabalha com Cookies, por isso é necessário tomar um pouco de cuidado com as informações que serão salvas, já que com apenas alguns cliques um usuário avançado conseguiria acesso às mesmas.

Na **Listagem 6**, você confere o código completo. O resultado final é exibido na **Figura 6**.

**Listagem 6**. Código completo da unit principal (projeto Cookies)

```
<?php
require_once("vcl/vcl.inc.php");
//Inclusões
use_unit("forms.inc.php");
use_unit("extctrls.inc.php");
use_unit("stdctrls.inc.php");

//Definição de classe
class Unit1 extends Page
{
        public $EdtValor = null;
        public $BtnRecuperar = null;
        public $BtnCriar = null;
        public $lblResultado = null;
        function BtnRecuperarClick($sender, $params)
        {
           $this->lblResultado->Caption = $_COOKIE["MyCookie"];
        }

        function BtnCriarClick($sender, $params)
        {
           setcookie("MyCookie", $this->EdtValor->Text, time()+3600);
        }

}

global $application;

global $Unit2;

//Cria o formulário
$Unit1=new Unit1($application);

//Ler do arquivo de recursos
$Unit1->loadResource(__FILE__);

//Mostrar o formulário
$Unit1->show();

?>
```

**Figura 6.** Resultado - Cookies

## JavaScript

Embora existam várias tecnologias como Flash, SilverLight entre outras utilizadas para a criação de aplicações RIA (Rich Internet Application) que a cada dia vêm ganhando força no mercado, JavaScript sempre será utilizado. A interatividade que temos com a parte cliente é sem dúvida o ponto forte do JavaScript.

No Delphi For PHP também é possível trabalhar com JavaScript. Nesta pequena dica, criaremos um simples exemplo, onde ao clicar em um Button será retornada uma mensagem utilizando a função alert do JavaScript. Crie um novo projeto e salve o mesmo com o nome de JavaScript, adicione um Button e no seu evento OnClick da guia JavaScript coloque o código a seguir:

```
alert('Mensagem com JavaScript');
```

Na **Listagem 7** você confere o código completo deste exemplo. O resultado é mostrado na **Figura 7**.

**Listagem 7**. Código completo da unit Principal (Projeto JavaScript)

```
<?php
require_once("vcl/vcl.inc.php");

//Inclusões
use_unit("forms.inc.php");
use_unit("extctrls.inc.php");
use_unit("stdctrls.inc.php");

//Definição de classe
class Unit1 extends Page
{
        public $Button1 = null;
        function Button1JSClick($sender, $params)
        {

        ?>
        alert('Mensagem com JavaScript');

        <?php
        }
}

global $application;

global $Unit2;

//Cria o formulário
$Unit1=new Unit1($application);

//Ler do arquivo de recursos
$Unit1->loadResource(__FILE__);

//Mostrar o formulário
$Unit1->show();

?>
```

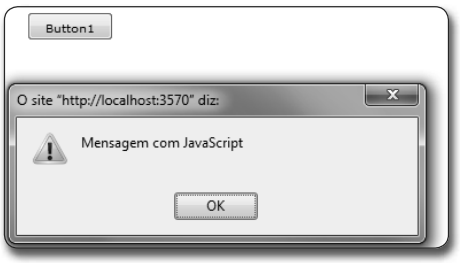



**Figura 7.** Resultado - JavaScript

## Conclusão

Este artigo foi prático e direto nas principais funcionalidades que com certeza serão utilizadas na maior parte de suas aplicações com Delphi For PHP. Vimos que o Delphi For PHP realiza com simplicidade, através de seus componentes, boa parte do trabalho que antes precisávamos realizar manualmente, digitando extensos códigos. Vimos que o Delphi For PHP é RAD para Web assim com o Delphi Win32 é RAD para Desktop. Aproveite seus recursos. Abraço e até o próximo artigo. ●

**Links**

**Download do Delphi For PHP**
*http://www.codegear.com/br/products/delphi/php*

**Download MySQL Server 5.1**
*http://dev.mysql.com/downloads/*

**Manual PHP**
*http://www.php.net/manual/pt_BR/*

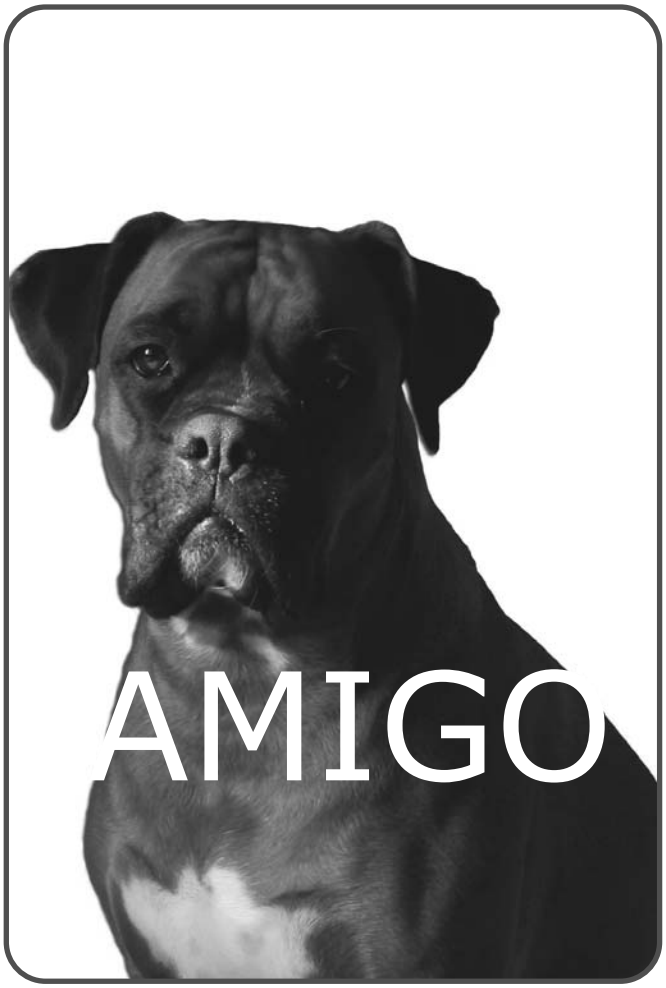
**Google Local Business Center**
*http://maps.google.com/local/add*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***




Existem coisas
que não conseguimos
ficar sem!

...só pra lembrar,
sua assinatura pode
estar acabando!

**Renove Já!**

www.devmedia.com.br/renovacao

Para mais informações:
www.devmedia.com.br/central

DevMedia GROUP

local
Data Center

Nossos **servidores** dedicados possuem monitoramento 24x7x365, link dedicado e **full duplex**, ou seja, os servidores atingem a **velocidade** do link tanto para **upload** quanto para **download.**

## HOST

Servidor
Sites
Gerenciamento

## DATACENTER

Co-location
E-mail
Backup

## TELECOM

Cross Connection
Banda IP
Link PAP

# SOLUÇÕES INTELIGENTES

Solicite um representante comercial

www.localdatacenter.com.br

SOLICITE UM ORÇAMENTO
TEL.: 3956 - 4900
comercial@localdatacenter.com.br

# Hospede seus sites de forma simples, rápida e ilimitada.

## Plano **Básico**

- 1 domínio principal
- 50 domínios hospedados
- Transferência **ilimitada**
- Espaço **ilimitado**
- 10 contas de e-mail
- PHP, ASP e ASP.NET
- Painel de controle Plesk
- Suporte 24hs

Tudo isso por apenas

R$ **15,90** por mês
pagamento trimestral

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

- Conheça também nossos planos profissionais com registro de domínio grátis e isenção de taxas anuais enquanto for cliente.

**HOSPEDIX**
HOSPEDAGEM PROFISSIONAL
**www.hospedix.com.br**

**30 DIAS GRÁTIS**

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

**RVSTMEDIA**